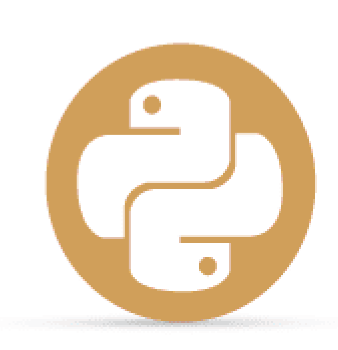


人工智能
在信用债投资领域的应用

Python语言实践

崔玉征◎著

清华大学出版社

人工智能在信用债投资领域的应用

Python语言实践

崔玉征　著

清华大学出版社

北　京

内 容 简 介

本书共分三部分,第一部分主要讲述机器学习、深度学习和人工智能的基本方法,并给出了使用基于 TensorFlow 后台的 Keras 库做深度学习的实践案例;第二部分主要讲述做信用债投资面临的困难,并给出了实用的解决方案;第三部分主要讲述解决做信用债投资的困难的实用方法,并给出了全部的 Python 源代码。

本书适合在银行、证券、保险、基金等金融机构从事对公信贷和债券投资等工作的相关从业者阅读。

图书在版编目(CIP)数据

人工智能在信用债投资领域的应用:Python 语言实践/崔玉征著. —北京:清华大学出版社,2019

ISBN 978-7-302-51305-6

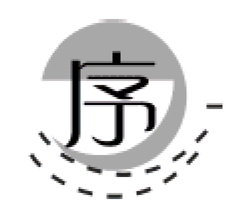
Ⅰ. ①人…　Ⅱ. ①崔…　Ⅲ. ①人工智能—应用—债券投资—研究　Ⅳ. ①F830.59-39

中国版本图书馆 CIP 数据核字(2018)第 234146 号

责任编辑: 张　伟
封面设计: 李召霞
责任校对: 宋玉莲
责任印制: 刘海龙

出版发行: 清华大学出版社
网　　址: http://www.tup.com.cn, http://www.wqbook.com
地　　址: 北京清华大学学研大厦 A 座　　**邮　　编:** 100084
社 总 机: 010-62770175　　**邮　　购:** 010-62786544
投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn
印 装 者: 三河市国英印务有限公司
经　　销: 全国新华书店
开　　本: 170mm×240mm　　**印　　张:** 14.75　　**字　　数:** 287 千字
版　　次: 2019 年 1 月第 1 版　　**印　　次:** 2019 年 1 月第1 次印刷
定　　价: 69.00 元

产品编号:081058-01



序

我的学生崔玉征先生，拥有中科院模式识别（人工智能）和香港中文大学金融 MBA 两个专业的硕士学位，是典型的复合型人才。他一直致力于通过机器学习等人工智能方法，解决国内资本市场信用债投资时所面临的外部评级虚高、无法真实有效地反映融资主体的信用状况、无法用于融资成本定价等行业痛点。

中国债券市场，自从 2014 年 3 月 5 日第一只信用债“超日债”违约以来，截至现在已经有 228 只信用债发生违约，且违约事件的发生有明显加速的迹象。随着中国经济进入中低速、但高质量增长的新常态，及未来很长一段时间我国可能面临的严峻外部经济环境，可以预判中国债券市场违约事件会频发，违约进入新常态。

美国著名经济学家、诺贝尔经济学奖得主保罗·克鲁格曼教授，提出了著名的“三元悖论”，即开放经济下的政策选择只能同时满足：本国货币政策的独立性、汇率的稳定性和资本的完全流动性中的两个，而无法同时满足全部三个。改革开放 40 年来，中国的宏观经济政策在货币政策相对独立、人民币汇率相对稳定的前提下，主要依靠人口红利、制度红利和资源红利等的不断释放，实现了快速的跨越式发展，发展中出现的问题被高速的经济增长所掩盖。但是，随着这些红利的不断消失，之前不断积累的问题也都已经慢慢地凸显出来了。在金融领域表现得最突出的问题是金融稳定、渐进性改革、定价体系缺失造成的套利机会并存，这三个矛盾构成了中国金融政策的“三元悖论”。也就是说，为了维护中国的金融稳定，政府机构选择了“修修补补”式的渐进性改革，而这种形式的渐进性改革并没有完全遵守金融具有全局性属性的基本原理，这就必然会对定价体系造成扭曲，从而产生大量的套利机会。大量套利机会的存在必然影响中国的金融稳定。因此，这是中国金融政策的“不可能三角”。

问题的存在一直以来都是我们发展的动力，中国经济要想实现高质量的健康发展，需要不断化解中国金融政策面临的“不可能三角”。伴随着人工智能、大数据和移动互联网等技术的不断发展，金融科技不断推动着传统金融业的变革，我们逐渐地看到了解决中国金融政策“不可能三角”的突破口。那就是，通过逐渐建立各层次、各类别资本市场的定价体系，来逐渐消除套利机会

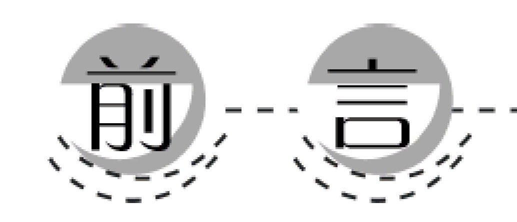
并建设稳定的金融体系。目前，债券市场已经初步具备了建设合理定价体系的条件，这主要是因为随着债券市场违约事件的不断增多，信用利差在不断扩大，尤其是高收益债券的信用利差出现明显扩大的趋势，这就为债券发行主体的定价提供了有利的前提条件。债券市场合理定价的前提是有一个市场化的信用评级体系，该体系可真实有效地反映债券发行主体信用风险的高低。

债券市场现有的外部评级体系，由于监管、财报粉饰较普遍等原因，表现出了明显的“虚高”和评级调整严重滞后的弊端。我认为本书提出的基于场外实时数据并采用机器学习技术的量化评级方法可较好地解决外部评级的弊端。这些场外数据主要包括司法、招聘、股权出质、动产抵押、高管变动、对外投资、实际控制人风险等，完全可以从非财务的侧面反映一家公司真实的经营状况，剔除了由于财务粉饰给我们造成的噪声干扰，仅保留能够真实反映企业还款能力的信号。这在国内资本市场的信用评级领域是一个较大的创新。我期待着这种量化研究方法逐渐得到业内同仁的认可。

何　佳

原证监会规划发展委员会委员、深交所综合研究所所长，南方科技大学教授

2018年9月25日



前言

笔者硕士毕业于中国科学院自动化研究所，专业是模式识别与智能系统。因此，人工智能是笔者的专业！ 10 年前笔者在自动化所读书时，人工智能这个专业远没有像今天这样火爆。 随着 AlphaGo 在人机博弈领域的围棋对弈中不断取得辉煌的成绩，人工智能再次走向辉煌，几乎被很多媒体描述成“无所不能”。

其实，人工智能有强弱之分，即强人工智能和弱人工智能。 强人工智能通常表现为在多个领域可用计算机来代替人类或者比人类做得更好，而弱人工智能通常是指在某一特定领域计算机比人类做得更好，如棋牌类游戏、量化投资等领域。 目前，强人工智能还只是幻想，弱人工智能在某些特定领域已经取得了惊人的成绩。 那么，人工智能可应用于哪些领域呢？ 通常地讲，能产生大量数据且这些数据可实现自动标注的领域，均适用于人工智能。 例如金融行业，它每天都产生大量的数据，且这些数据都可实现自动标注，因为股票要么上涨、要么下跌，债券要么正常兑付、要么违约。 因此，无场景不 AI（人工智能）。

深度学习技术的突破是这次人工智能浪潮的巨大推动力之一。 深度学习的思想跟传统机器学习相比，是一个颠覆性创新，在思维方式上是完全不同的。传统机器学习一般是通过大量数据寻找因果关系，而深度学习一般是通过分层特征提取并通过激活函数寻找关联关系，这正是大数据方法的思维方式。

本书重点介绍的是人工智能方法在信用债投资领域应用的实践方法，提供了大量的 Python 源代码，可供信用债投资者快速建立自己的信用债量化投资信用风险分析体系。 随着国内债券市场的快速发展，目前存量债券已经超过 35 000 只，如此多的投资标的，如果还是采用传统的、基于人工的主观判断和信息筛选，这个工作量是非常巨大的。 自从 2014 年 3 月 5 日国内第一只债券“超日债”违约以来，截至现在已经有 228 只债券发生违约，信用风险越来越受到投资者的重视，建设自己的、能够为投资决策提供有效支持的信用风险计量和管理体系，对广大金融机构来说已经非常迫切。

但是，建设科学、实用的信用债投资分析体系，在国内资本市场的现状下面临很多无法解决的客观困难。 例如，发债主体信息披露不及时、财务粉饰现象普遍存在、违约状态跟盈利状况正相关、违约样本极少，造成样本极不均衡、外部评级过度集中且区分能力很差等现状，作为普通的信用债投资者我们均无法

改变这些现状。我们只能从优化模型架构设计、获取可提供稳定场外数据源且实用的指标等角度入手，来建立自己的信用债投资量化风险分析体系。

本书介绍的信用债投资量化风险分析体系是笔者近10年来经验的总结和升华，书中详细介绍了通过模型架构优化来缓释财报粉饰的模型开发方法，介绍了通过场外真实数据来开发模型的技术，还简单介绍了深度学习技术在信用债投资量化风险分析体系中的应用方法。这些方法既是笔者多年经验的结晶，也是市面上其他同类教材中从来没有介绍过的实用技术总结。

由于财务粉饰现象的普遍存在，基于财务数据的企业信用风险评估经常出现误判，无法挖掘企业真正的投资价值。本书介绍的全部基于企业真实定性数据的信用风险评估方法已经得到了笔者团队的充分验证，效果是非常显著的。这种方法体系和实践方法在国内首屈一指，笔者坚信会给信用债投资分析者很大的启发。

本书共分为八个章节，其中第1章、第2章重点介绍人工智能、机器学习、深度学习的基本理论和方法，也给出了国内资本市场最常见的类别不均衡问题的解决方案；第3章重点介绍基于TensorFlow后台的Keras深度学习架构，并给出实用案例；第4章重点介绍国内债券市场的发展历程和现状；第5章重点介绍国内信用债投资分析面临的困难，并给出了详细的解决方案；第6～8章是本书的开源技术部分，详细讲述了大量的核心技术，包括自动抓取数据、对全市场财务数据的统计检验和分析、对司法等场外数据的统计检验和分析、财务粉饰的本福特统计法则和评分实施方法、用有监督机器学习开发评级模型的方法、用深度学习技术开发评级模型的方法、缓释财报粉饰的评分卡模型架构设计和评分卡模型开发核心代码等，这三章是本书开源技术的核心部分。

在本书的写作过程中，得到了家人、朋友和团队成员的大力支持，没有他们的支持和帮助，笔者不可能心无旁骛地构思本书的写作思路，在此对他们的贡献表示真诚的感谢。本书的前五章和附录部分由笔者完成，第6～8章中的部分Python源代码由团队成员刘志兴等完成初稿，并由笔者完成测试和剩余的逻辑分析部分。

读者若有问题与作者交流，请发邮件至yuzheng.cui@qq.com。扫一扫以下二维码，可获得本书的源代码。



崔玉征

2018年5月8日


目
录

的一系列问题，且不能被辨别出其机器身份，则该计算机通过测试，说明该计算机具备智能。图灵测试的基本原型，如图 1.1 所示。

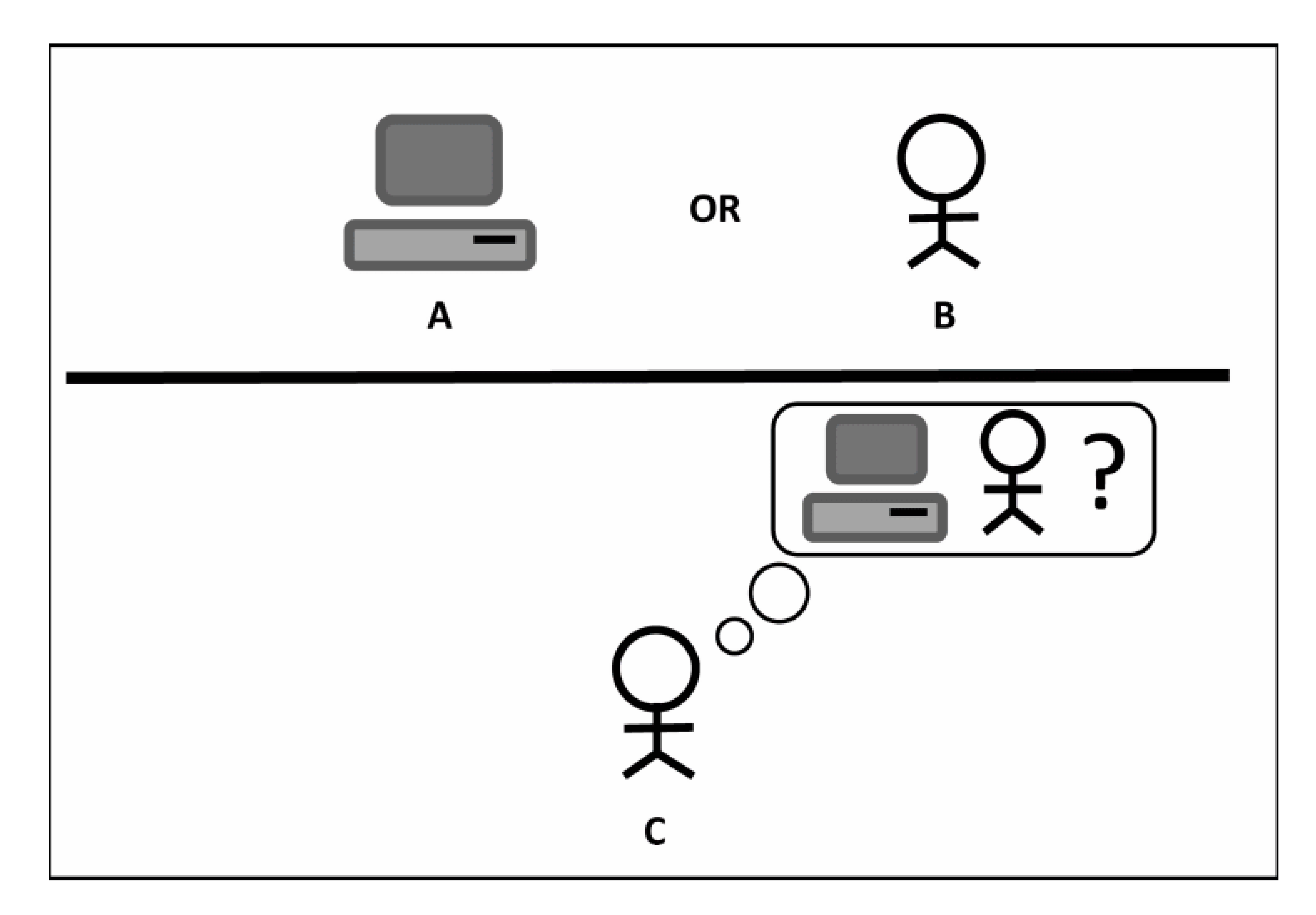


图 1.1 图灵测试的基本原型

在图 1.1 所示的图灵测试的基本原型中，C 代表人类，他通过一定的媒介向被测试者 A 或 B 提问一系列的问题，通过测试者对这些问题的回答，如果 C 无法辨别回答者是人还是机器，则说明被测试者具有智能。

图灵还为这项测试拟定了几个示范性问题，如下所示。

问：请给我写出有关“第四号桥”主题的十四行诗。

答：不要问我这道题，我从来不会写诗。

问：34 957 加 70 764 等于多少？

答：(停 30 秒后)105 721。

问：你会下国际象棋吗？

答：是的。

问：我在 K1 处有棋子 K；你在 K6 处有棋子 K，在 R1 处有棋子 R。轮到你走，你应该下哪步棋？

答：(停 15 秒钟后)棋子 R 走到 R8 处，将军！

由图灵测试的基本原型可知，在某一特定细分领域，即提问者 C 提出的问题仅限于某一特定的狭窄领域，此时通过大量的数据测试和模型训练，是不难通过图灵测试的。这种应用于细分领域的人工智能通常被称为弱人工智能。红极一时的 AlphaGo 就属于弱人工智能的典型应用，也就是说 AlphaGo 只能用于下围棋，不能直接用于其他领域。如果想将 AlphaGo 应用于其他领域，需要用该领域的数据做大量训练后，才有可能适用。

如果提问者 C 提出的问题，不预设场景、可随意提问，被测试者仍能通过图灵测试，则这时通常被称为强人工智能。显然，强人工智能基本是不可能的。因

为这需要机器存储人类有史以来的所有数据，并训练出人类所有可能的知识和推理，这是根本不可能完成的工作。

可见，人工智能是机器智能，不是人类智能，也不是类人智能。机器智能是必须通过提取大量数据，并进行自动标注后计算得到的智能。而除此之外，人类智能还包括不可计算的智能，如情感、直觉、感知等。

1.2 人工智能、机器学习和深度学习

由图灵测试的基本原型可知，人工智能不是一项技术，而是一种技术结果的最终展现。即一系列技术的成果通过了图灵测试，则认为该技术成果具备了智能。实现人工智能有很多种方法，包括规划调度、专家系统、多代理系统、进化计算、机器学习、知识表述、推荐系统等，如图 1.2 所示。

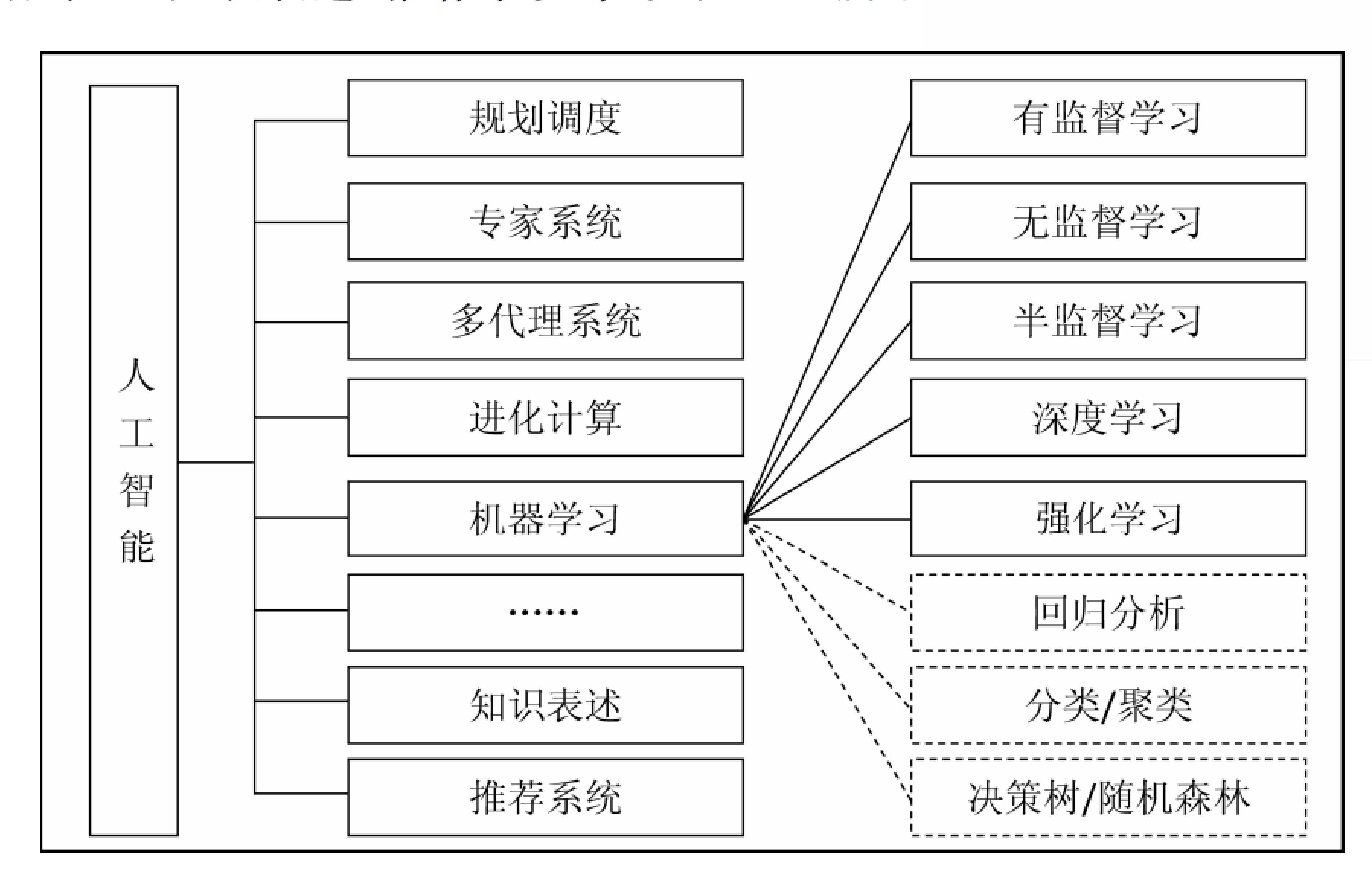


图 1.2 部分人工智能研究分支和研究方法

目前，用机器学习技术实现人工智能的方法产生了令人兴奋的结果。机器学习是一大类技术的统称，又可分为有监督机器学习技术（需要标注数据）和无监督机器学习技术（不需要标注数据），而深度学习又是有监督机器学习技术中的一个重要研究分支。人工智能、机器学习和深度学习的区别与联系如下所示。

1. 机器学习：一种实现人工智能的方法

机器学习最基本的做法，是使用算法来解析数据、从数据中按照一定的规则训练出规律，然后对真实世界中的事件作出决策和预测。与传统的为解决特定任务、硬编码的软件程序不同，机器学习是用大量的数据来“训练”，通过各种算法从数据中学习如何完成任务。举个简单的例子，当我们浏览网站时，经常会出

现电子商务网站广告的商品推荐信息。这是电子商务网站根据你以往的购物、点击和浏览记录，识别出这其中哪些是你真正感兴趣，并且愿意购买的产品，并推荐给你。这样的决策模型，可以帮助电子商务网站为客户提供建议并鼓励购买产品。机器学习直接来源于早期的人工智能领域，传统的算法包括决策树、K 均值聚类、贝叶斯分类、支持向量机等。从学习方法上来分，机器学习算法可以分为有监督学习（如分类问题）、无监督学习（如聚类问题）、半监督学习、集成学习、深度学习和强化学习等。传统的机器学习算法在指纹识别、人脸检测、语音识别等领域的应用基本达到了商业化的要求或者特定场景的商业化水平，但每前进一步都异常艰难，直到深度学习算法的出现。

2. 深度学习：一种实现机器学习的技术

深度学习本来并不是一种独立的机器学习方法，其本身也会用到有监督和无监督的机器学习技术来训练深度神经网络模型。但由于近几年该领域发展迅猛，一些特有的学习手段相继被提出（如残差网络），因此越来越多的人将其单独看作一种机器学习的方法。

最初的深度学习是利用深度神经网络来解决特征表达的一种学习过程。深度神经网络本身并不是一个全新的概念，可大致理解为包含多个隐含层的神经网络结构。为了增强深层神经网络的训练效果，人们对神经元的连接方法和激活函数等方面作出相应的调整。其实有不少想法早年间也曾有过，但由于当时训练数据量不足、计算能力落后，因此最终的效果不尽如人意。目前，深度学习令人惊讶地实现了很多任务，使得似乎所有的机器辅助功能都变为可能，如无人驾驶、预防性医疗保健，甚至是更好的电影推荐，都近在眼前，或者即将实现。

3. 三者的区别和联系

机器学习是一种实现人工智能的方法，深度学习是一种实现机器学习的技术。我们就用最简单的方法，可视化地展现出它们的关系，如图 1.3 所示。

当下深度学习在计算机视觉、自然语言处理领域的应用远超过传统的机器学习方法，并且媒体对深度学习进行了大肆夸大的报道。因此，目前业界有一种错误的、较为普遍的认识，那就是“深度学习最终可能会淘汰掉其他所有机器学习算法”。深度学习，作为目前最热门的机器学习方法，但并不是机器学习的终点。其主要存在以下问题。

（1）深度学习模型需要大量的训练数据，才能表现出良好的效果，但现实生活中往往会遇到小样本问题，此时深度学习方法表现差强人意，传统的机器学习方法就可以很好地处理。

（2）有些领域，采用传统的简单机器学习方法，就可以很好地解决问题了，没必要非得用复杂的深度学习方法。

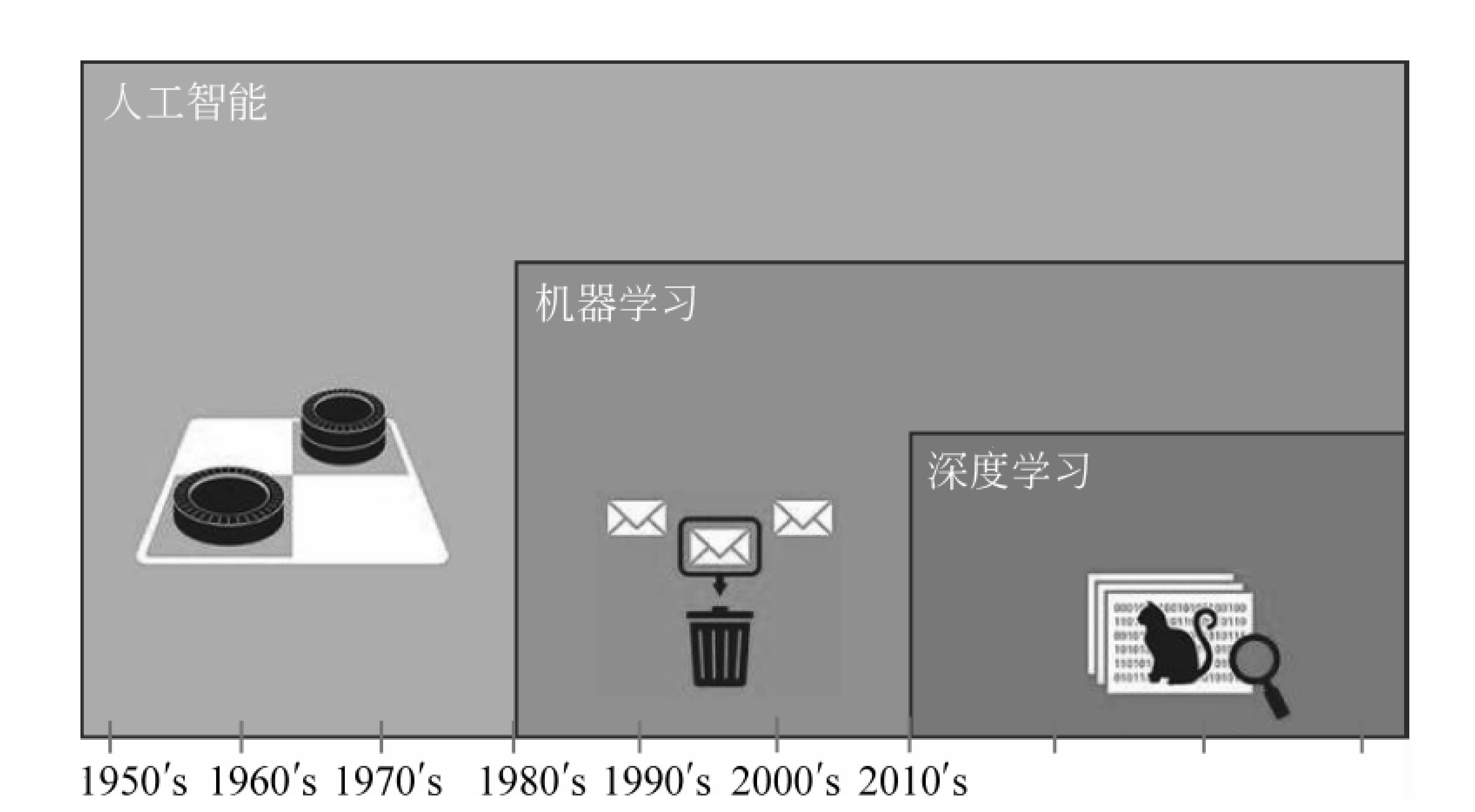


图 1.3　人工智能、机器学习和深度学习的关系

(3) 深度学习的思想,来源于人脑的启发,但绝不是人脑的模拟。举个例子,给一个 4 岁的小孩看一辆自行车之后,再见到哪怕外观完全不同的自行车,小孩也十有八九能作出那是一辆自行车的判断。也就是说,人类的学习过程往往不需要大规模的训练数据,而现在的深度学习方法显然不是对人脑的模拟。



第 2 章 机器学习

为了更好地了解机器学习，我们首先举个简单的例子，说明人的学习过程。夏天时，我们都去超市买过西瓜，通常通过观察、拍打等方法来找到成熟的甜瓜。观察主要是看西瓜外表的色泽和根蒂的形状，拍打主要是听声音。随着我们经验的不断积累，采用上述方法挑选出“好瓜”的概率越来越大。我们经验不断积累的过程，可用表 2.1 所示的数据集表示。

表 2.1　西瓜数据集

序号	色泽	根蒂	声响	是否好瓜
1	青绿	干缩	浊响	是
2	黑绿	干缩	浊响	是
3	青绿	硬挺	清脆	否
4	黑绿	硬挺	沉闷	否
…	……	……	……	……

通过人脑的不断归纳、总结，我们可以得出“好瓜”的模型是“某种色泽、某种根蒂、某种声响的瓜”。人脑的归纳和总结，就是人的学习过程，这是一个非常复杂的、非线性的神经元学习网络。

那么，机器的学习过程是怎样的呢？我们知道，计算机只能处理数字信号，且只能判断确定的规则，如大于、小于、不等于等。对于模糊的规则，如感知、感觉等则无法判断。举个简单的例子说明机器的学习过程，如下所示。

问：1＋4 等于几？

答：20。

规则反馈：不对，多了。

问：1＋9 等于几？

答：13。

规则反馈：不对，多了。

问：3＋4 等于几？

答：7。

规则反馈：对了。

问：6＋9等于几？

答：13。

规则反馈：不对，少了。

很多很多次以后……

问：2＋2等于几？

答：4。

问：4＋5等于几？

答：9。

这就是机器的学习过程，准确来说这是机器学习方法中最常用的一种，叫有监督机器学习。“监督”的意思，就是数据集训练中的规则反馈。

最开始的几步是对模型的训练，“多了”或“少了”可以理解为训练时的误差监督反馈，模型根据误差调整自身参数，这就是机器学习里常用的反向传播（back propagation）的简单解释。

通过以上示例分析，并结合专业教材对机器学习的定义，此处，我们给出最直观、最易理解的定义，机器学习是一种实现人工智能的科学，主要是通过数据可自动总结规律并改进性能的计算机算法的研究。

2.1 机器学习概述

通常，根据收集的样本是否存在标注，将机器学习分为有监督机器学习和无监督机器学习两类。有监督机器学习所需要的样本集数据，不仅仅包括特征数据，还包括每个样本的一个准确输出值，该输出值即为对该样本的标注。如果收集的样本中没有对每个样本的标注，则该数据集只能用于无监督机器学习。例如，如果我们只收集每只股票的开盘价、收盘价、成交额、成交量、换手率数据，则该数据集只能用于无监督机器学习；而如果我们除此之外还收集每只股票的涨跌信息，并将涨跌信息用于预测，则该数据集适用于有监督机器学习。

2.1.1 有监督机器学习

有监督机器学习中的预测结果可以是连续值，也可以是离散值。根据这样的属性可将有监督机器学习分为回归问题（regression）和分类问题（classification）。回归问题预测的结果一般为连续值，即因变量为数量变量。分类问题预测的结果一般为离散值，即因变量为分类变量。

1. 回归问题

回归问题主要是通过回归算法来探索样本数据与标注数据之间关系的一类问题，而衡量回归算法的规则通常是误差最小。用图2.1所示的回归问题示例

图表示，回归问题就是使用计算机算法寻找一条直线（$Y=aX+b$），使得图中的每个红点距离该直线的偏差之和最小。

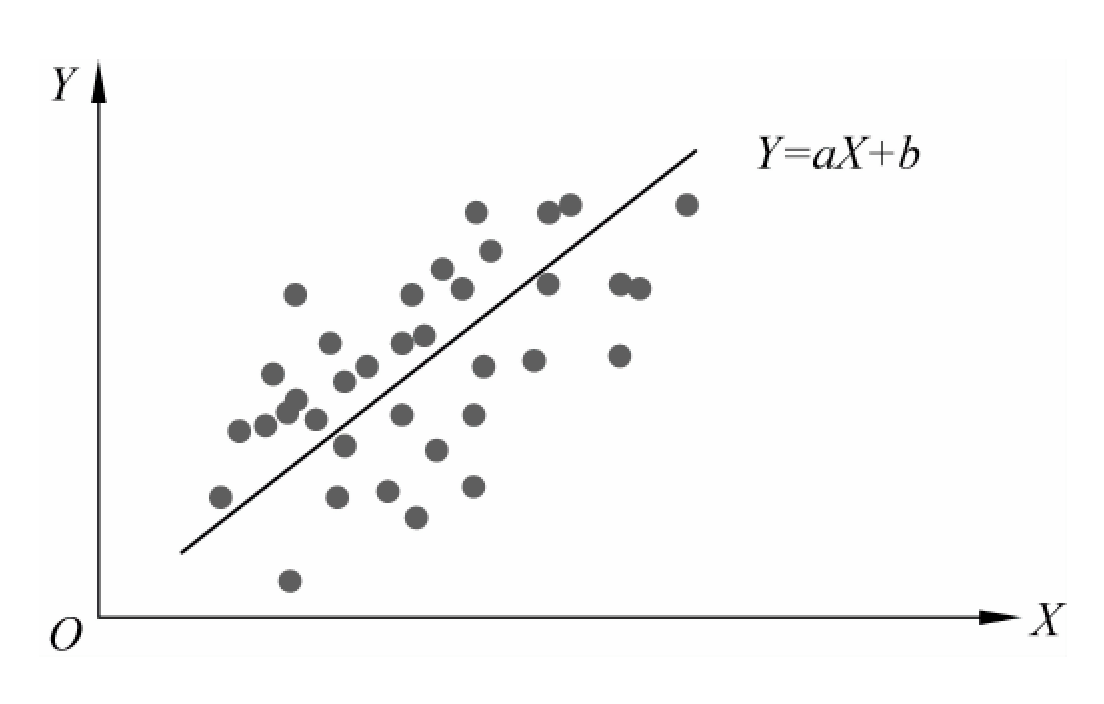


图 2.1　回归问题示例图

在有监督机器学习算法中，常见的回归问题算法有线性回归、逻辑回归、决策树回归、随机森林回归、支持向量机回归等。此处，我们分别以简单的示例说明这些算法的使用方法。

```
#回归问题算法采用的公用数据集
from sklearn import datasets, preprocessing
from sklearn.model_selection import train_test_split
iris=datasets.load_iris()
X, y=iris.data, iris.target
X_train, X_test, y_train, y_test=train_test_split(X, y, random_state
=33)                                                        #随机抽样
scaler=preprocessing.StandardScaler().fit(X_train)          #标准化数据
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
#1. 线性回归
from sklearn.linear_model import LinearRegression
lr=LinearRegression(normalize=True)
lr.fit(X, y)                            #采用线性回归算法训练模型
y_pred=lr.predict(X_test)               #用线性回归模型预测
#2. 逻辑回归
from sklearn.linear_model import LogisticRegression
lg=LogisticRegression()
lg.fit(X,y)                             #采用逻辑回归算法训练模型
y_pred=lg.predict(X_test)               #用逻辑回归模型预测
#3. 决策树回归
from sklearn.tree import DecisionTreeRegressor
dtr=DecisionTreeRegressor(max_depth=3)
dtr.fit(X,y)                            #采用决策树回归算法训练模型
y_pred=dtr.predict(X_test)              #用决策树回归模型预测
```

```
#4. 随机森林回归
from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor(max_depth=2, random_state=0)
rfr.fit(X, y)                          #采用随机森林回归算法训练模型
y_pred=rfr.predict(X_test)             #用随机森林回归模型预测
#5. 支持向量机回归
from sklearn import svm
clf=svm.SVR()
clf.fit(X, y)                          #采用支持向量机回归算法训练模型
y_pred=clf.predict(X_test)             #用支持向量机回归模型预测
```

2. 分类问题

分类问题主要是通过分类算法来根据已知样本的某些特征，判断一个新的样本属于哪种已知的样本类别，而衡量分类算法的规则通常包括准确率、召回率等。用图 2.2 所示的分类问题示例图表示，分类问题就是使用计算机算法寻找一个判别函数，使得尽可能准确地将图中的两类样本区分开来。

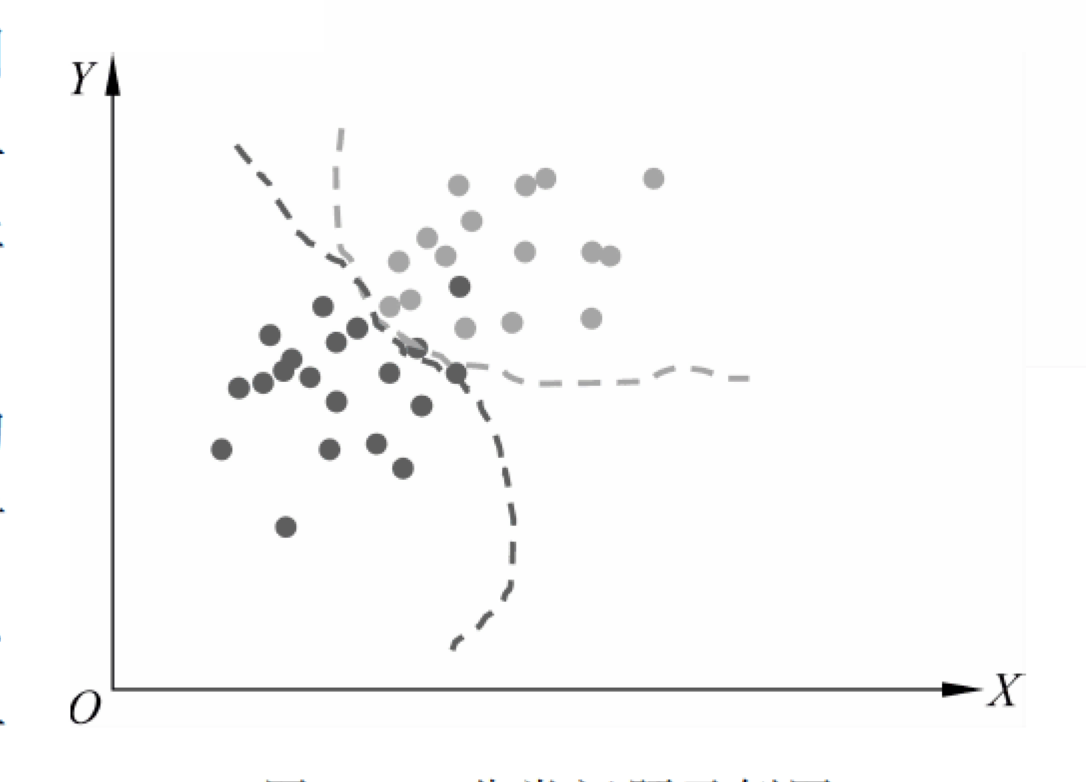


图 2.2　分类问题示例图

在有监督机器学习算法中，常见的分类问题算法有逻辑回归分类、决策树分类、随机森林分类、支持向量机分类、最近邻分类等。此处，我们分别以简单的示例说明这些算法的使用方法。

```
#公用数据集
from sklearn import datasets, preprocessing
from sklearn.model_selection import train_test_split
iris=datasets.load_iris()
X, y=iris.data,iris.target
X_train, X_test, y_train, y_test=train_test_split(X, y, random_state
=33)
scaler=preprocessing.StandardScaler().fit(X_train)    #标准化数据
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
#1. 逻辑回归分类
from sklearn.linear_model import LogisticRegression
lg_clf=LogisticRegression()
lg_clf.fit(X,y)                          #用逻辑回归分类算法训练模型
y_pred=lg_clf.predict(X_test)            #用逻辑回归分类模型预测
#2. 决策树分类
```

```
from sklearn.tree import DecisionTreeClassifier
dt_clf=DecisionTreeClassifier(random_state=0)
dt_clf.fit(X,y)                                  #用决策树分类算法训练模型
y_pred=dt_clf.predict(X_test)                    #用决策树分类模型预测
#3. 随机森林分类
from sklearn.ensemble import RandomForestClassifier
rf_clf=RandomForestClassifier(max_depth=2, random_state=0)
rf_clf.fit(X,y)                                  #用随机森林分类算法训练模型
y_pred=rf_clf.predict(X_test)                    #用随机森林分类模型预测
#4. 支持向量机分类
from sklearn import svm
svm_clf=svm.SVC()
svm_clf.fit(X, y)                                #用支持向量机分类算法训练模型
y_pred=svm_clf.predict(X_test)                   #用支持向量机分类模型预测
#5. k近邻(kNN)分类
from sklearn.neighbors import KNeighborsClassifier
neigh=KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)                                  #用kNN分类算法训练模型
y_pred=neigh.predict(X_test)                     #用kNN分类模型预测
```

2.1.2 无监督机器学习

无监督机器学习中,不需要对收集的样本进行标注,也不需要知道学习的结果是否正确。无监督机器学习的目的是对原始资料进行分类,以便了解资料的内部结构。在无监督机器学习中,最典型的例子就是聚类。聚类的目的是把相似的东西聚在一起,而我们并不关心这一类是什么。图 2.3 简单展示了聚类问题的一般结果。

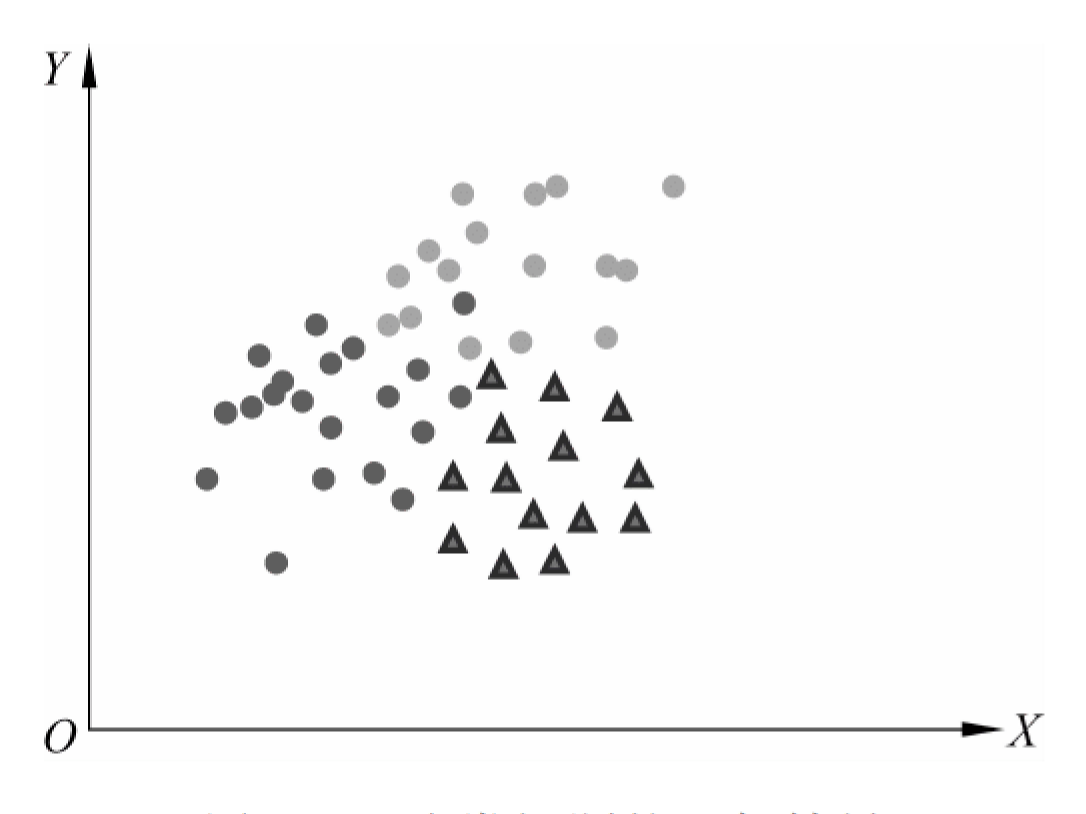


图 2.3 聚类问题的一般结果

在无监督机器学习算法中,常见的算法有 K 均值聚类、主成分分析等。此处,我们分别以简单的示例说明这些算法的使用方法。

```
from sklearn import datasets
iris=datasets.load_iris()
X=iris.data
y=iris.target
#1. KMeans 聚类
from sklearn.cluster import KMeans
est=KMeans(n_clusters=8)
est.fit(X)                          #使用 KMeans 算法进行聚类
print(est.labels_)                  #输出类别的标签
#2. PCA
from sklearn import decomposition
pca=decomposition.PCA()
pca.fit(X)                          #使用 PCA 算法进行主成分分析
print(pca.explained_variance_)      #输出每个指标的解释方差,以便于计
                                     算累积贡献率
```

2.1.3 半监督机器学习

有监督机器学习的样本需要有标注,且标注数据质量的好坏直接影响机器学习的效果(包括回归问题和分类问题)。无监督机器学习的样本不需要标注,但这类学习算法只能用于了解样本结构,而不能用于预测。在现实问题中,如果样本量过大,全部标注是件非常困难的事情。经常遇到只有部分标注样本,但需要处理回归或分类问题的情景,此时就需要用到半监督机器学习技术,图 2.4 所示为半监督机器学习示意图。

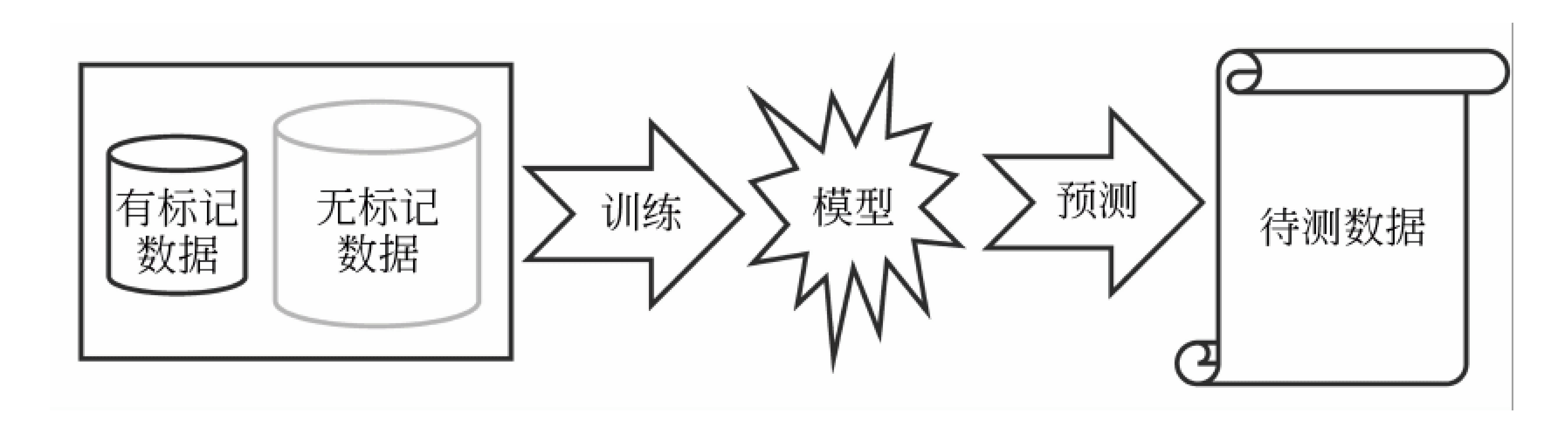


图 2.4　半监督机器学习示意图

所谓半监督机器学习技术,就是指让学习器不依赖外界交互、自动地利用未标注样本来提升学习性能的一类算法。在 sklearn. semi_supervised 包中实现了基于标注传播(Label Propagation)原理的半监督机器学习算法。此处,我们以简单的示例说明标注传播算法的使用方法。

```
import numpy as np
from sklearn import datasets
from sklearn.semi_supervised import label_propagation
rng=np.random.RandomState(0)
```

```
iris=datasets.load_iris()
X=iris.data[:, :2]
y=iris.target
y_30=np.copy(y)
y_30[rng.rand(len(y))<0.3]=-1     #去掉已有的标注(随机数小于 0.3 的索引)
y_50=np.copy(y)
y_50[rng.rand(len(y))<0.5]=-1     #去掉已有的标注(随机数小于 0.5 的索引)
ls30=label_propagation.LabelSpreading(kernel='knn', alpha=0.8)
ls30.fit(X, y_30)                 #训练模型
ls50=label_propagation.LabelSpreading(kernel='knn', alpha=0.8)
ls50.fit(X, y_50)                 #训练模型
predicted_labels_ls30=ls30.transduction_      #输出预测的标注
predicted_labels_ls50=ls50.transduction_      #输出预测的标注
```

本节简单介绍了三类机器学习技术的基本原理，分别是有监督机器学习、无监督机器学习和半监督机器学习，并给出了使用相关算法的简单示例。为了便于梳理、理解机器学习的技术体系，我们用图 2.5 所示的分类图给本节作一个总结。

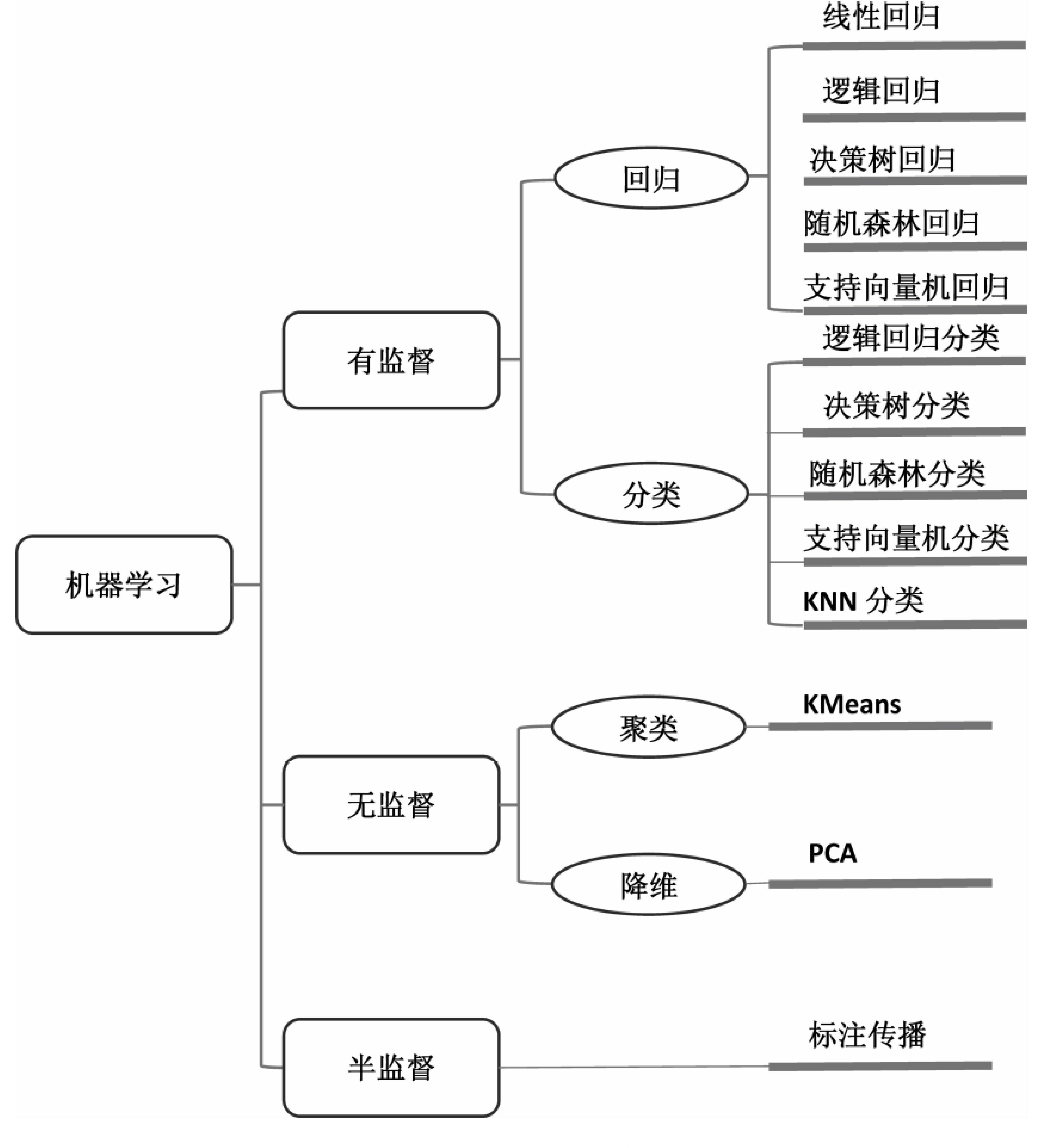


图 2.5　机器学习技术概述

2.2 深度学习

深度学习是机器学习的一个重要分支，其用到的核心技术是人工神经网络。当提到深度学习的概念时，很多初学者会问：一个机器学习模型中，使用几层神经网络才能视为深度学习模型呢？这个问题看似有道理，实际上并没有深入理解深度学习的基本原理，而是将深度学习看成神经网络层次的简单叠加。实际上，深度学习模型和一般神经网络模型相比在原理上有很大不同，这主要表现在特征提取和处理的顺序与方式上。深度学习处理问题的顺序和方式跟人类思考问题的方式比较接近。例如，深度学习在图像分类问题中的一般流程是首先提取图片的基础特征——图片像素，然后将图片像素特征输入神经网络的第一层，并提取线条特征，紧接着将线条特征输入到神经网络的第二层，提取图片中物体的简单形状特征，最后将简单形状特征输入神经网络的第三层，提取图片中物体的复杂形状特征。然后再将提取的这些多层复杂特征进行权重学习，并将得到的模型用于图片分类预测。深度学习和传统机器学习方法处理问题的一般流程，如图 2.6 所示。

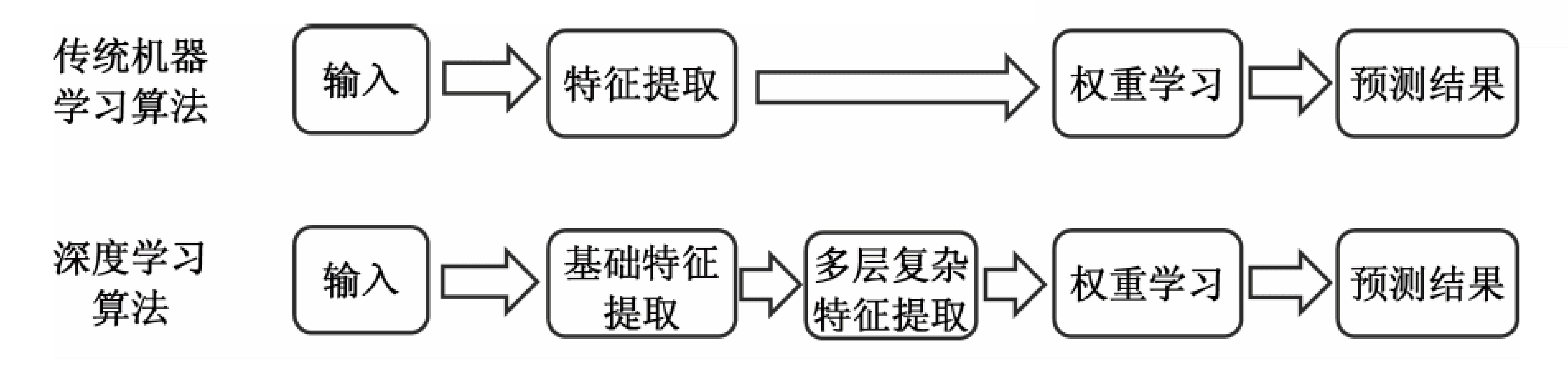


图 2.6　深度学习和传统机器学习方法处理问题的一般流程

显然，图片分类问题深度学习的过程，和人类学习绘画的过程比较类似。幼儿学习绘画时，首先认识颜料，然后学习简笔画，最后学习复杂绘画并涂色。那么，什么是深度学习呢？简单地说，如果模型使用分层特征学习——首先识别较低级别的特征，然后建立在它们之上以识别更高级的特征（如通过卷积滤波器等），那么它就是一个深度学习模型。如果没有分层特征学习，那么无论你的模型有多少层，都不是真正的深度学习模型。即我们不是因为深度模型（多层神经网络模型）而将其称为深度学习，而是为了实现层次化学习，模型需要深度，深度是实现分层特征学习的副产品。因此，深度学习是使用多层神经网络技术来实现分层特征学习的一种算法。

为了实现深度学习模型中的分层特征学习，我们需要用到神经网络的模型结构，神经网络最基本的结构单元称神经元，这是由生物学引出的一个概念。生物学中一个神经元通常具有多个树突，主要用来接收传入的信息；而轴突只有一条，轴突尾端有许多轴突末梢可以给其他多个神经元传递信息；轴突末梢和其他

神经元的树突产生连接，从而传递信息。为了模拟生物神经元的结构和信息传递方式，人工神经网络中的神经元结构也基本类似，如图 2.7 所示。

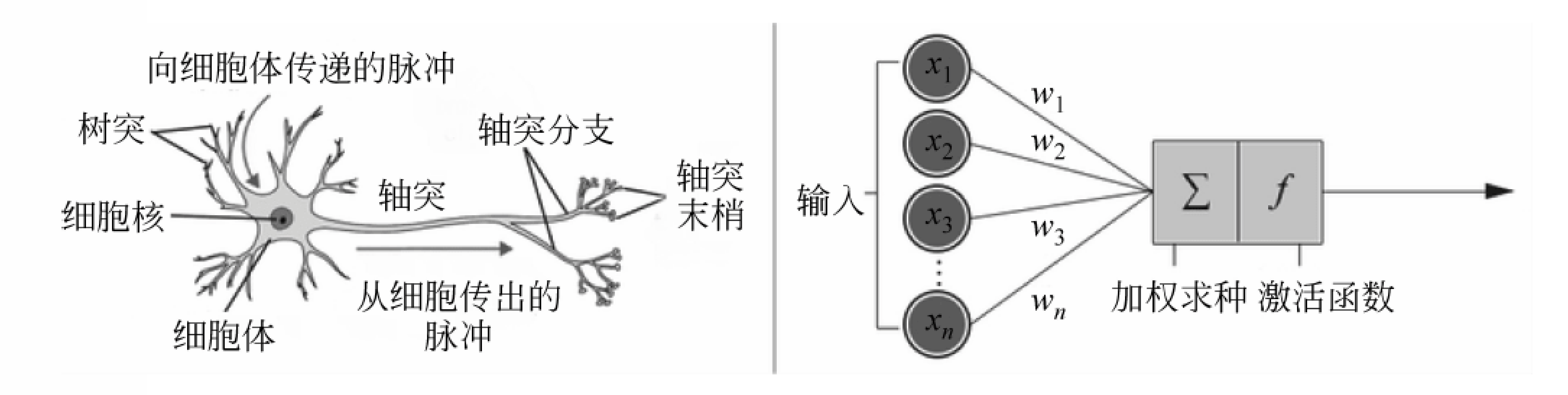


图 2.7　生物神经元与人工神经网络结构对比图

在如图 2.7 右侧所示的人工神经元中，输入为 $x_1, x_2, \cdots, x_n$，每个输入的权重为 $w_1, w_2, \cdots, w_n$，将这些输入的加权求和作为激活函数的输入，并计算后得到该神经元的输出，即该神经元的输出为 $y = f(w_1 \times x_1 + w_2 \times x_2 + \cdots + w_n \times x_n)$。显然，各输入节点的加权求和是一个线性表达式，为处理非线性的分类问题，激活函数通常选择非线性函数，这样训练得到的深度学习模型就可以处理非线性分类问题了。训练深度学习模型的过程，就是计算每个人工神经元输入的权重的过程。

随着神经网络层次的增多和每个神经元激活函数的不同，训练得到的神经网络模型处理复杂问题的能力一般也会逐步提升，如图 2.8 所示。

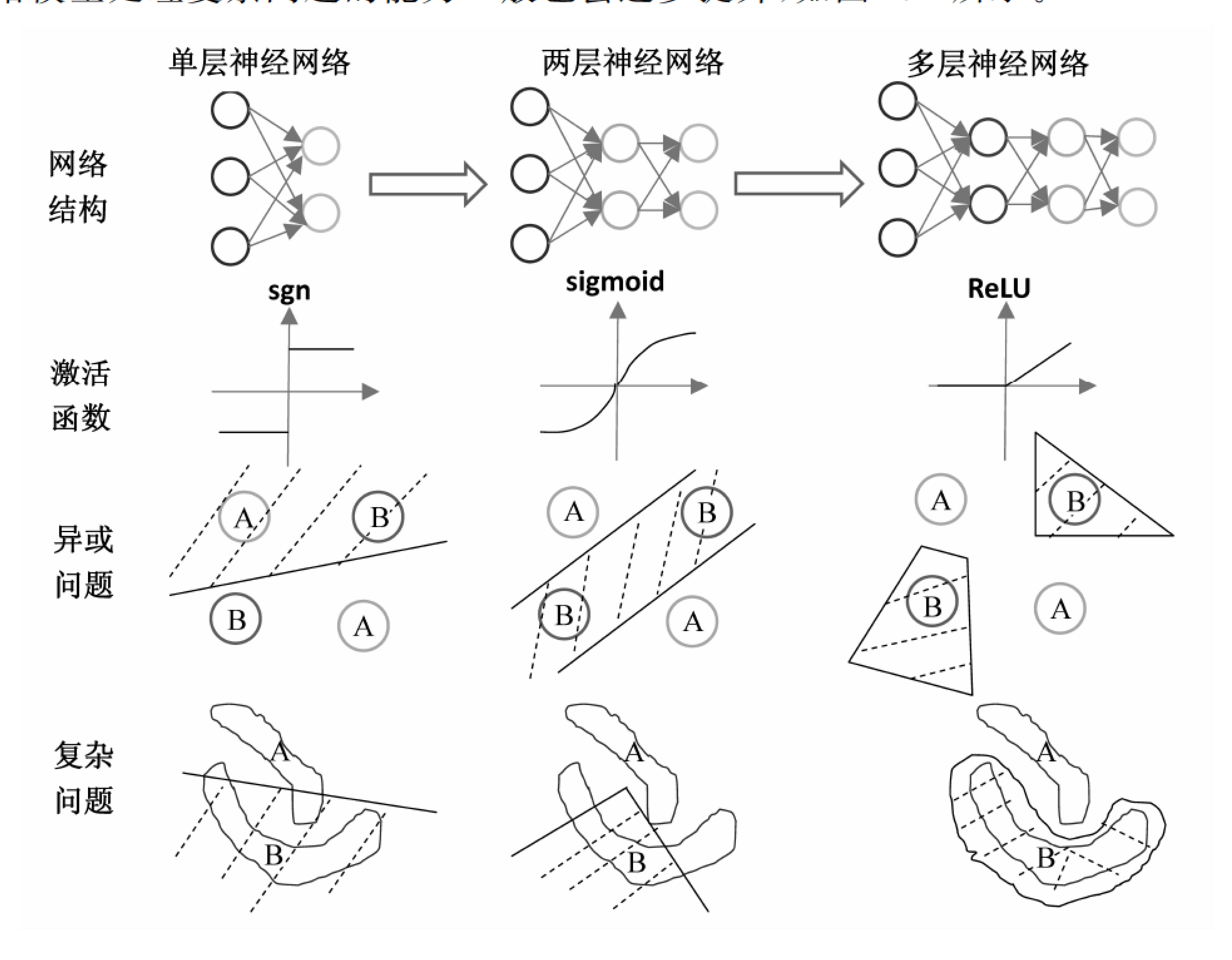


图 2.8　神经网络分类能力不断增强

此处，我们以一个具体的实例来说明深度神经网络的构建、训练和验证过程。为了进行深度学习，本书采用 Keras 深度学习库（关于 Keras 的安装、配置与用法请见本书第 3 章的详细介绍）。

```
#需要指出的是，该深度学习代码只能在 Linux 平台下并安装 Keras 深度学习库后，才能正常运行。详细的安装与配置方法，请见 3.2 节中的详细介绍。
#载入所需要的包
import pandas as pd
from keras.datasets import fashion_mnist
import numpy as np
from keras.utils import to_categorical
import matplotlib.pyplot as plt
%matplotlib inline
#获取测试数据，包括训练集和测试集
(train_X,train_Y),(test_X,test_Y)=fashion_mnist.load_data()#使用 Keras 内置函数下载数据
#由于数据量较大、运行时间较长，读者也可在开源平台 GitHub 自行下载数据集，网址为：https://github.com/zalandoresearch/fashion-mnist，如图 2.9 所示，单击 Download 即可下载程序运行所需数据集。
```


Get the Data

Many ML libraries already include Fashion-MNIST data/API, give it a try!

You can use direct links to download the dataset. The data is stored in the **same** format as the original MNIST data.

Name	Content	Examples	Size	Link	MD5 Checksum
train-images-idx3-ubyte.gz	training set images	60,000	26 MBytes	Download	8d4fb7e6c68d591d4c3dfef9ec88bf0d
train-labels-idx1-ubyte.gz	training set labels	60,000	29 KBytes	Download	25c81989df183df01b3e8a0aad5dffbe
t10k-images-idx3-ubyte.gz	test set images	10,000	4.3 MBytes	Download	bef4ecab320f06d8554ea6380940ec79
t10k-labels-idx1-ubyte.gz	test set labels	10,000	5.1 KBytes	Download	bb300cfdad3c16e7a12a480ee83cd310

图 2.9 Fashion-MNIST 训练样本

```
print('Training data shape : ', train_X.shape, train_Y.shape)#输出训练样本的结构，有 60 000 个训练样本，每个样本均由 28×28 的数组构成
print('Testing data shape : ', test_X.shape, test_Y.shape)#输出测试样本的结构，有 10 000 个测试样本，每个样本均由 28×28 的数组构成
#获取训练样本的类别
classes=np.unique(train_Y)
nClasses=len(classes)
print('Total number of outputs : ', nClasses)#输出类别，共计 10 类
print('Output classes : ', classes)#输出类别，分别用 0~9 的数字表示，如
```

表 2.2 所示。

表 2.2　样本的类别

类别标记	描　　述	类别标记	描　　述
0	T-shirt/top	5	Sandal
1	Trouser	6	Shirt
2	Pullover	7	Sneaker
3	Dress	8	Bag
4	Coat	9	Ankle boot

```
#查看训练和测试样本的第一幅图,如图 2.10 所示
plt.figure(figsize=[5,5])
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Class Number: {}".format(train_Y[0]))
plt.subplot(122)
plt.imshow(test_X[0,:,:], cmap='gray')
plt.title("Class Number: {}".format(test_Y[0]))
```

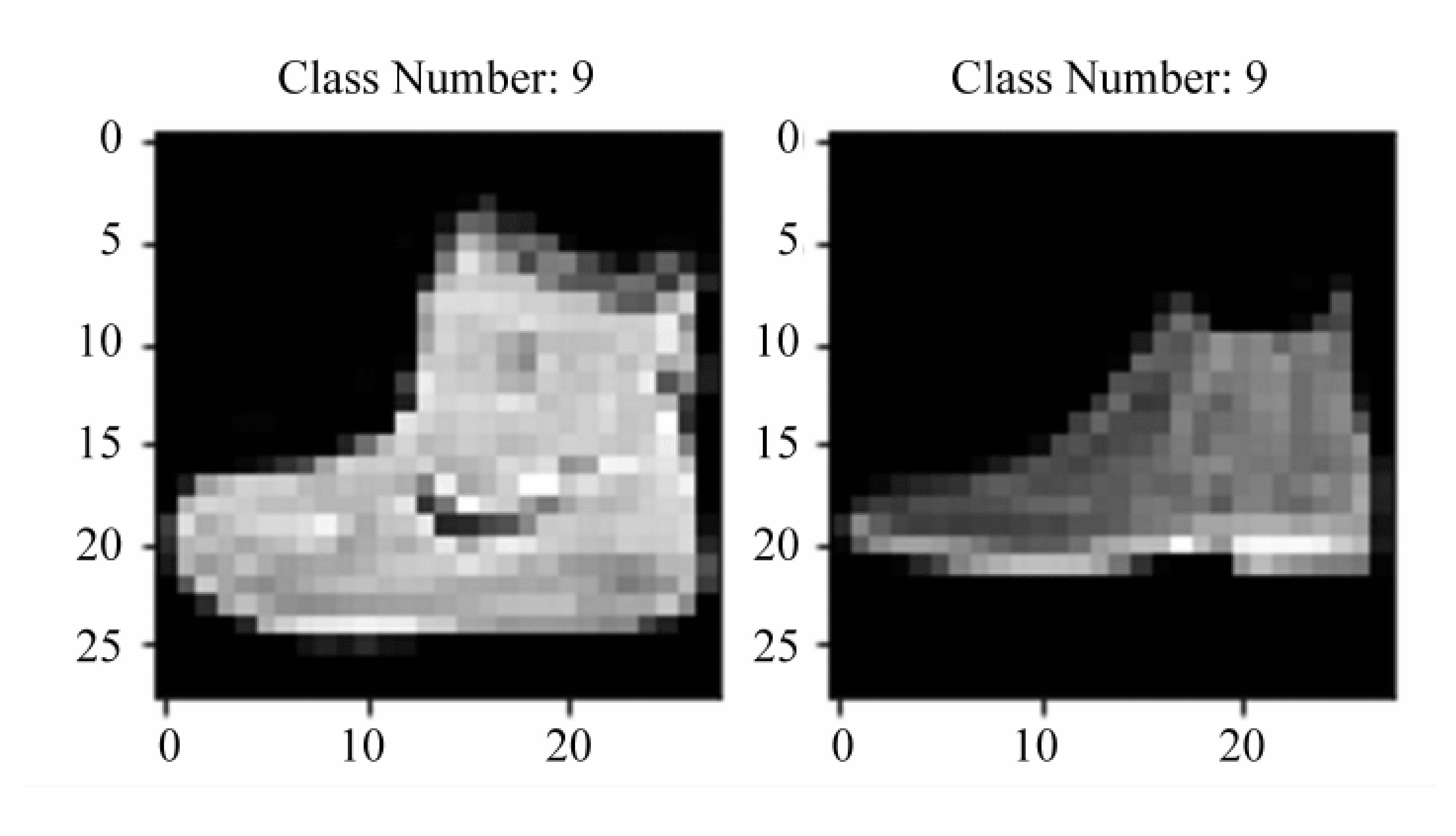


图 2.10　训练和测试样本的第一幅图

```
#训练和测试样本均为灰度图,将它们转化为 28×28×1 的矩阵存储
train_X=train_X.reshape(-1, 28,28, 1)
test_X=test_X.reshape(-1, 28,28, 1)
train_X.shape, test_X.shape #查看样本转换后的 shape,输出结果为((60 000,
28, 28, 1), (10 000, 28, 28, 1))
#原样本中存储的数据类型为整数,需要转换为浮点数,并做归一化处理
train_X=train_X.astype('float32')
test_X=test_X.astype('float32')
train_X=train_X / 255.
```

```
test_X=test_X / 255.
#由于机器学习算法不能直接处理元类别(0~9),因此需要进行1位热编码(one-hot enconding)。例如类别9的1位热编码为[0 0 0 0 0 0 0 0 1 0],即用一个布尔型数组,只有第9位为1,其他都为0。
train_Y_one_hot=to_categorical(train_Y)
test_Y_one_hot=to_categorical(test_Y)
#查看将类别(0~9)进行1位热编码转换前后的变化
print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
#将训练样本集分为训练集和测试集,训练集用于训练模型,测试集用于检验模型。此处将原样本集中的80%用于训练模型,20%用于测试模型
from sklearn.model_selection import train_test_split
train_X,valid_X,train_label,valid_label=train_test_split(train_X,
train_Y_one_hot, test_size=0.2, random_state=13)
#检查训练集和测试集的shape
train_X.shape,valid_X.shape,train_label.shape,valid_label.shape
#加载构建深度学习模型所需要的包
import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU
batch_size=64 #进行梯度下降时每个batch包含的样本数,训练时一个batch的样本会被计算一次梯度下降,使目标函数优化一步
epochs=20 #模型训练次数
num_classes=10#使用Keras中的序列模型(Sequential)构建卷积神经网络
fashion_model=Sequential()
fashion_model.add(Conv2D(32,kernel_size=(3,3),activation='linear',
input_shape=(28,28,1),padding='same'))#向序列模型中添加第一层二维卷积层,卷积核的数目为32(输出的维度为32维),卷积核的大小为3×3,激活函数为linear,输入数据的shape为28×28×1,same代表保留边界处的卷积结果
fashion_model.add(LeakyReLU(alpha=0.1)) #为第1层卷积神经网络的输出添加激活函数
fashion_model.add(MaxPooling2D((2, 2),padding='same'))#二维最大值池化,使图片在两个维度上均变为原长的一半
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding=
'same')) #向序列模型中添加第2层二维卷积层,卷积核的数目为64(输出的维度为64维),卷积核的大小为3×3
fashion_model.add(LeakyReLU(alpha=0.1)) #激活函数
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#二维最大值池化,使图片在两个维度上均变为原长的一半
```

```
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same')) #向序列模型中添加第 3 层二维卷积层,卷积核的数目为 128(输出的维度为 128 维),卷积核的大小为 3×3
fashion_model.add(LeakyReLU(alpha=0.1)) #激活函数
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#二维最大值池化,使图片在两个维度上均变为原长的一半
fashion_model.add(Flatten())#用于将输入"压平",即把多维的输入一维化,用于卷积层到全连接层的过渡
fashion_model.add(Dense(128, activation='linear'))#全连接层,激活函数为 linear
fashion_model.add(LeakyReLU(alpha=0.1)) #激活函数
fashion_model.add(Dense(num_classes, activation='softmax'))#全连接层,使得输出数据维数跟样本类别维数相同(此处为 num_classes=10)
fashion_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adam(),metrics=['accuracy'])#编译模型,深度学习网络构建完成后,需要编译后才能运行
fashion_model.summary()#查看深度学习模型的结构,如图 2.11 所示。
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 32)	320
leaky_re_lu_1 (LeakyReLU)	(None, 28, 28, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496
leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 7, 7, 64)	0
conv2d_3 (Conv2D)	(None, 7, 7, 128)	73856
leaky_re_lu_3 (LeakyReLU)	(None, 7, 7, 128)	0
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 128)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 128)	262272
leaky_re_lu_4 (LeakyReLU)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

Total params: 356,234
Trainable params: 356,234
Non-trainable params: 0

图 2.11　深度学习模型的结构

```
fashion_train=fashion_model.fit(train_X,train_label,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,valid_label))#训练神经网络模型,共计训练 20 次
test_eval=fashion_model.evaluate(test_X, test_Y_one_hot, verbose=
```

```
0)#计算模型误差
print('Test loss:', test_eval[0])#输出测试误差结果丢失率 loss(44.46%)
print('Test accuracy:', test_eval[1])#输出测试准确率 accuracy(91.54%)
#绘制准确率和丢失率曲线,以判断模型是否过拟合,如图 2.12 所示。
```

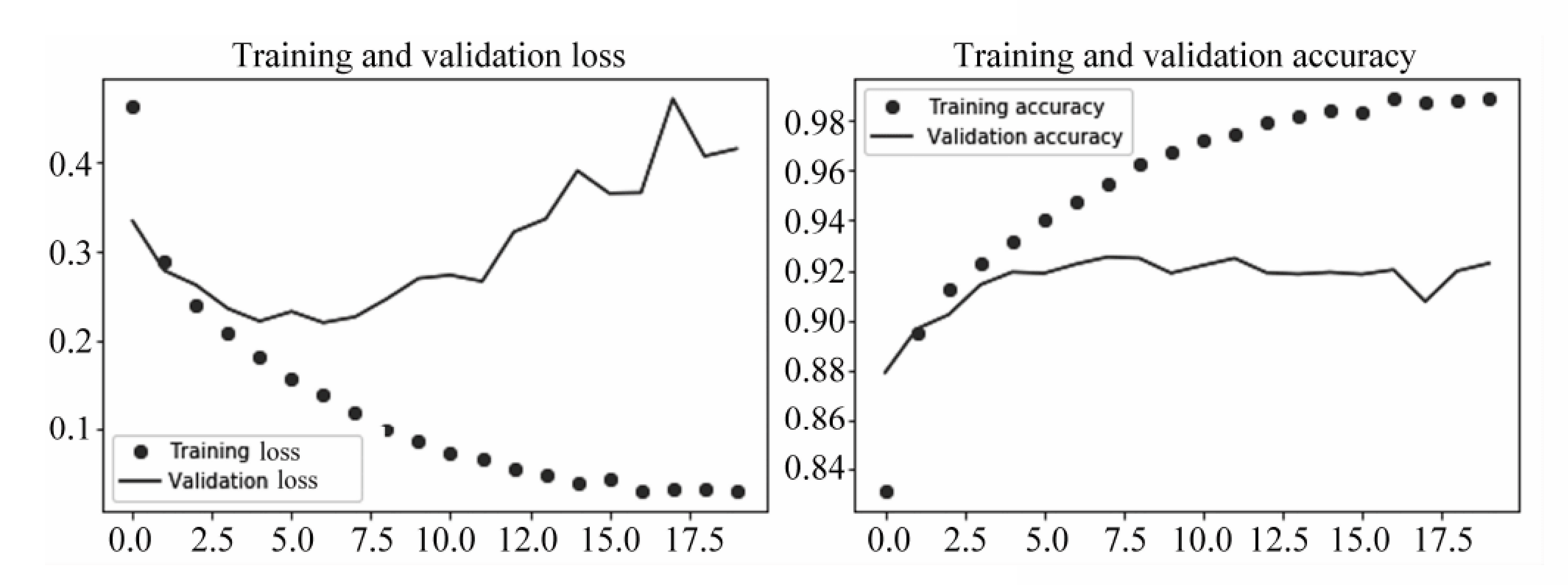


图 2.12　训练和测试样本的 loss/accuracy 曲线

```
#图 2.12 中横坐标表示训练次数,从中可以看出随着训练次数的不断增加训练样本的丢失率不断减少、准确率不断提高,而测试样本却出现了相反的结果。这表明,训练得到的模型出现了过拟合现象。
accuracy=fashion_train.history['acc']
val_accuracy=fashion_train.history['val_acc']
loss=fashion_train.history['loss']
val_loss=fashion_train.history['val_loss']
epochs=range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
#为了防止模型出现过拟合现象,我们需要在神经网络的每一层后添加 Dropout 层,这样即可有效降低过拟合现象。
batch_size=64
epochs=20
num_classes=10
fashion_model=Sequential()
fashion_model.add(Conv2D(32,kernel_size=(3,3),activation='linear',
padding='same',input_shape=(28,28,1)))
```

```
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D((2, 2),padding='same'))
fashion_model.add(Dropout(0.25))#添加 Dropout 层,使得每次训练模型更新参数时,按 25%的概率随机断开输入神经元
fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.25))#添加 Dropout 层,断开输入神经元的概率为 25%
fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
fashion_model.add(Dropout(0.4)) #添加 Dropout 层,断开输入神经元的概率为 40%
fashion_model.add(Flatten())
fashion_model.add(Dense(128, activation='linear'))
fashion_model.add(LeakyReLU(alpha=0.1))
fashion_model.add(Dropout(0.3))#添加 Dropout 层,断开输入神经元的概率为 30%
fashion_model.add(Dense(num_classes, activation='softmax'))
fashion_model.summary()#查看模型结构
fashion_model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])#编译模型
fashion_train_dropout=fashion_model.fit(train_X,train_label,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X, valid_label))#训练模型,参数保持不变
fashion_model.save("fashion_model_dropout.h5py")#保存模型
test_eval=fashion_model.evaluate(test_X, test_Y_one_hot, verbose=1) #计算模型误差
print('Test loss:', test_eval[0]) #输出测试误差结果丢失率 loss(22.74%)
print('Test accuracy:', test_eval[1]) #输出测试准确率 accuracy(91.95%)
#输出添加 Dropout 层后,训练和测试样本的 loss/accuracy 曲线,如图 2.13 所示。从图中我们可以看出随着训练次数的增加,训练集和测试集的准确率同步增加,丢失率同步减少。因此模型分类效果很好,且没有出现过拟合现象
accuracy=fashion_train_dropout.history['acc']
val_accuracy=fashion_train_dropout.history['val_acc']
loss=fashion_train_dropout.history['loss']
val_loss=fashion_train_dropout.history['val_loss']
epochs=range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
```

图 2.13　添加 Dropout 层后训练和测试样本的 loss/accuracy 曲线

```
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
predicted_classes=fashion_model.predict(test_X)#用训练得到的模型,对测试样本做分类预测
predicted_classes=np.argmax(np.round(predicted_classes),axis=1)
predicted_classes.shape, test_Y.shape#共计预测了 10 000 个样本的分类情况
correct=np.where(predicted_classes==test_Y)[0]#统计分类正确的情况
print("Found %d correct labels" %len(correct))#输出分类正确的个数(9 165 个)
#绘出 9 个分类正确的图片,如图 2.14 所示。
```

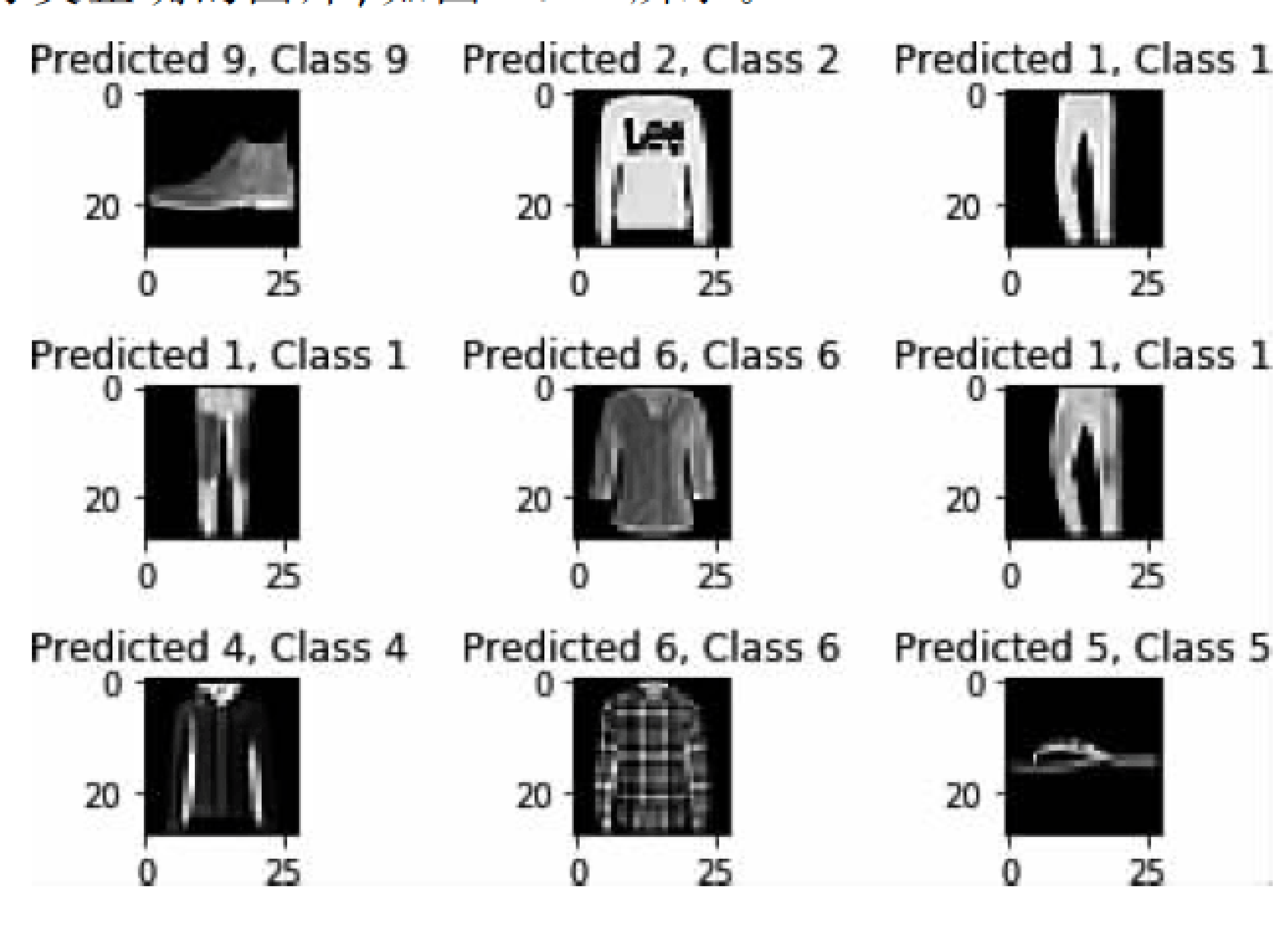


图 2.14　分类正确的图片

```
for i, correct in enumerate(correct[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow (test _ X [correct]. reshape (28, 28), cmap = ' gray ',
    interpolation='none')
    plt.title (" Predicted { }, Class { }". format (predicted _ classes
    [correct], test_Y[correct]))
    plt.tight_layout()
incorrect=np.where(predicted_classes! =test_Y)[0] #统计分类错误的情况
print("Found %d incorrect labels" %len(incorrect)) #输出分类错误的个数
(835 个)
#绘出 9 个分类错误的图片,如图 2.15 所示。
```

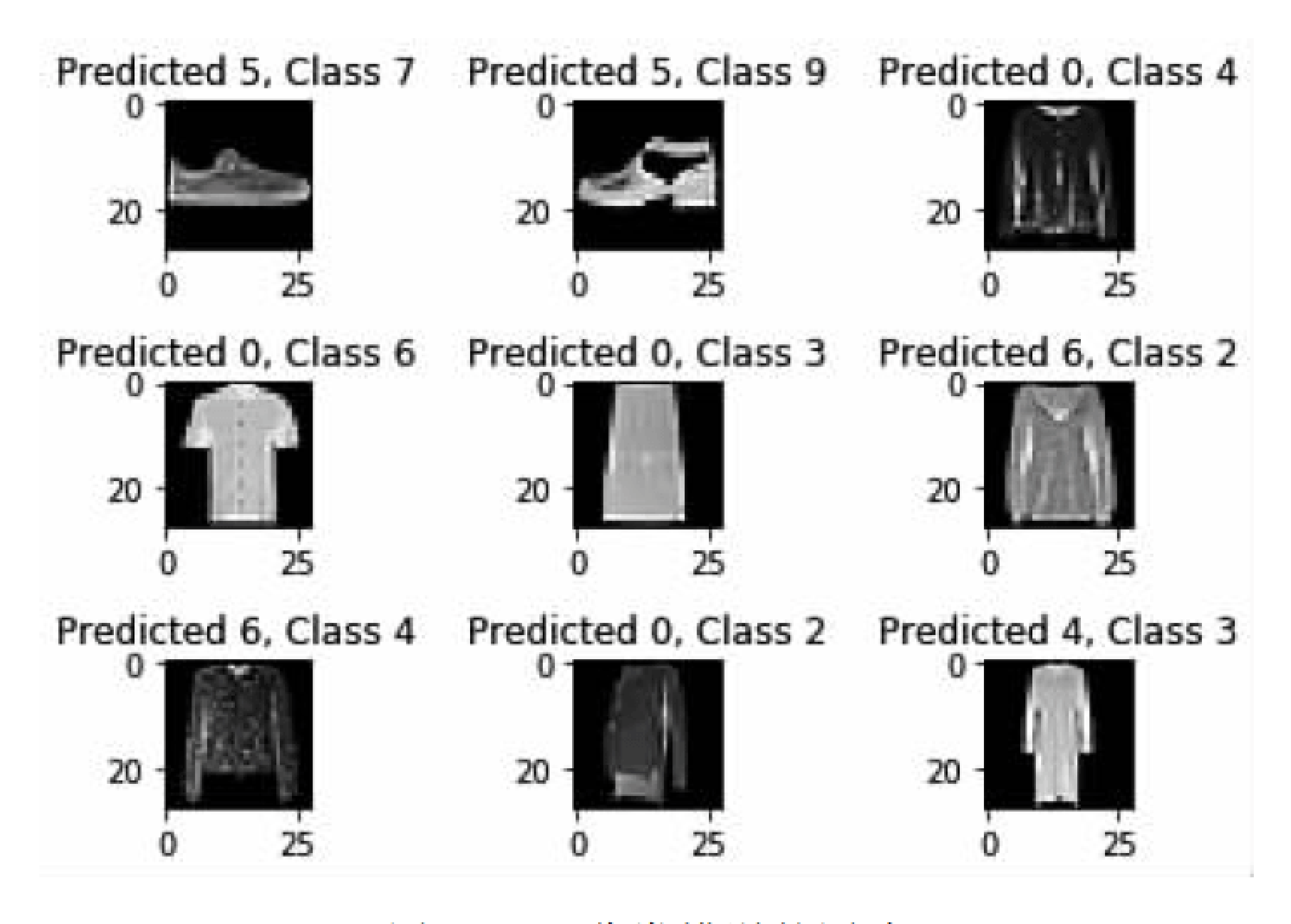


图 2.15　分类错误的图片

```
for i, incorrect in enumerate(incorrect[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow (test _ X [incorrect]. reshape (28, 28), cmap = ' gray ',
    interpolation='none')
    plt.title("Predicted{},Class{}".format(predicted_classes
        [incorrect], test_Y[incorrect]))
    plt.tight_layout()
#输出分类预测汇总情况,如图 2.16 所示。
from sklearn.metrics import classification_report
target_names=["Class {}".format(i) for i in range(num_classes)]
print(classification_report(test_Y,predicted_classes,target_names
=target_names))
```

	precision	recall	f1-score	support
Class 0	0.78	0.90	0.84	1000
Class 1	0.99	0.98	0.99	1000
Class 2	0.90	0.86	0.88	1000
Class 3	0.91	0.92	0.92	1000
Class 4	0.86	0.89	0.88	1000
Class 5	0.98	0.99	0.99	1000
Class 6	0.81	0.70	0.75	1000
Class 7	0.97	0.97	0.97	1000
Class 8	0.99	0.97	0.98	1000
Class 9	0.97	0.97	0.97	1000
avg / total	0.92	0.92	0.92	10000

图 2.16　分类预测汇总情况

2.3　类别不均衡问题的解决方案及 Python 源代码

本章前述的机器学习任务中都有一个共同的基本假设，即不同类别的训练样本数量相当。如果不同类别的训练样本数量差别不大，通常影响较小，但如果差别很大，则会对学习过程造成困扰。例如在 1 000 个训练样本中，有 998 个正常，只有 2 个违约，那么学习方法只需要返回一个永远将新样本预测为正常的学习器，就能达到 99.8%的精度。然而，这样的学习器往往没有价值，因为它不能预测出任何违约样本。这种样本总体中类别差距较大的情况，称为类别不均衡问题。

简单的线性分类器可很容易解释这种现象。在我们用 $y=w^{\mathrm{T}}x+b$ 分类模型对新样本 x 进行分类时，实际上是用预测出的 y 值与一个阈值进行比较，如通常在 $y>0.5$ 时判别为违约样本，否则为正常样本。y 实际上表达了违约的可能性，比率 $\frac{y}{1-y}$ 则反映了违约可能性与正常可能性的比值，阈值设置为 0.5 恰表明分类器认为正常样本和违约样本的可能性相同，即分类器决策规则为

$$\text{如果}\frac{y}{1-y}>1\text{，则预测为违约} \tag{2.1}$$

然而，当训练集中正常、违约的样本量不同时，我们令 m^+ 表示正常的样本量，m^- 表示违约的样本量，则观测比率为 $\frac{m^-}{m^+}$。由于我们通常假设训练集是样本总体的无偏估计，因此该观测比率就代表了真实比率。于是，只要分类器的预测概率大于观测比率就应该判定为违约，即

$$\text{如果}\frac{y}{1-y}>\frac{m^-}{m^+}\text{，则预测为违约} \tag{2.2}$$

但是，我们的分类器是基于式(2.1)进行决策的。因此，需要对其预测值进行调整，使其在基于式(2.1)决策时，实际是在执行式(2.2)的决策。要做到这一点很容易，只需令

$$\frac{y'}{1-y'} = \frac{y}{1-y} \times \frac{m^+}{m^-} \tag{2.3}$$

这就是处理类别不均衡学习的一个基本策略——“再缩放”(rescaling),亦称“再平衡”(rebalance)。

再缩放的思想虽简单,但实际操作却并不平凡,主要是因为“训练集是真实样本总体的无偏估计”这个假设不成立了,也就是说,我们未必能有效地基于训练集观测比率来推断出真实的正常与违约的比率。现有技术上大体有三类做法:第一类是直接对训练集中的正常样本进行“欠采样”(undersampling),即去除一些正常样本使得正常、违约样本数量接近,然后再进行机器学习;第二类是对训练集中的违约样本进行“过采样”(oversampling),即增加一些违约样本使得正常、违约样本数量接近,然后再进行机器学习;第三类是直接基于原始训练集进行机器学习,但在用训练好的分类器进行预测时,将式(2.3)嵌入决策过程中,称为“阈值移动”(threshold-moving)。

欠采样法的时间开销通常远小于过采样法,因为前者丢弃了很多正常的样本,使得分类器训练集远小于初始训练集;而过采样法增加了很多违约样本,其训练集大于初始训练集。需要注意的是,过采样法不能简单地对初始违约样本进行重复采样,否则会导致严重的过拟合。过采样法的代表性算法 SMOTE 是通过对训练集中的违约样本进行插值来产生额外的违约样本。同样,欠采样法也不能随意丢弃正常样本,因为这样可能会丢失一些重要信息。欠采样法的代表性算法 EasyEnsemble 是利用集成学习机制,将正常样本划分为若干个集合供不同学习器使用,这样对每个学习器都进行了欠采样,但在全局来看却不会丢失重要信息。

Python 语言中提供了处理不均衡样本的包 imbalanced－learn,在使用前需要首先安装它,安装步骤如下所示。

第一步,打开 Anaconda Prompt。单击 Windows 开始按钮,在程序中找到 Anaconda 文件夹,如图 2.17 所示,单击 Anaconda Prompt 即可打开。

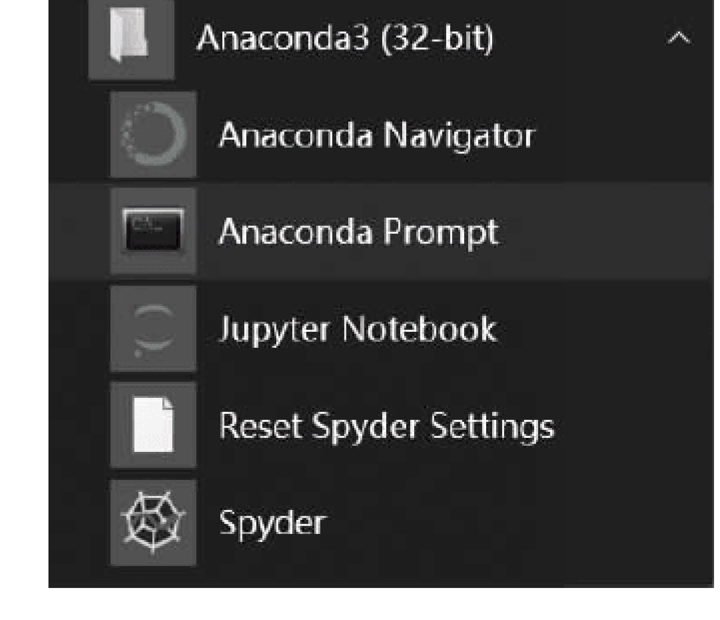


图 2.17　打开 Anaconda Prompt

在弹出的 Anaconda 命令提示符窗口,输入“conda install -c glemaitre imbalanced-learn”,并按回车键,然后单击 y 并再次按回车,即可安装 imbalanced-learn 包,如图 2.18 所示。

安装完成后,即可用该包处理类别不均衡样本的机器学习问题。此处,我们给出如下的应用示例,以演示过采样、欠采样等方法的处理步骤。

```
from sklearn.datasets import make_classification
X, y = make _ classification (n _ samples = 5000, n _ features = 2, n _
```

```
conda install -c glemaitre imbalanced-learn

(base) C:\Users\cuiyu>conda install -c glemaitre imbalanced-learn
Solving environment: done

## Package Plan ##

  environment location: D:\Anaconda

  added / updated specs:
    - imbalanced-learn


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    imbalanced-learn-0.3.1     |   pyh2cb239c_0          75 KB  glemaitre

The following NEW packages will be INSTALLED:

    imbalanced-learn: 0.3.1-pyh2cb239c_0 glemaitre

Proceed ([y]/n)? y
```

图 2.18　安装 imbalanced-learn 包

```
informative=2,
n_redundant=0, n_repeated=0, n_classes=3,n_clusters_per_class=1,
weights=[0.01, 0.05, 0.94],class_sep=0.8, random_state=0)
          #产生 5000 个样本,共计有 0/1/2 三个类别,用于处理机器学习的分类问题
#过采样算法
#1. 随机过采样
from imblearn.over_sampling import RandomOverSampler
ros=RandomOverSampler(random_state=0)      #随机过采样
X_resampled, y_resampled=ros.fit_sample(X, y)
from collections import Counter
print(sorted(Counter(y_resampled).items()))
                    #为每个类别分别产生相同的样本,数量为原始类别中最大的数量
#2. SMOTE 法过采样
from imblearn.over_sampling import SMOTE, ADASYN
X_resampled, y_resampled=SMOTE().fit_sample(X, y)  #使用 SMOTE 方法过采样
print(sorted(Counter(y_resampled).items()))        #输出过采样后的样本维
                                                    度,并按类别排序
#3. ADASYN 法过采样
X_resampled, y_resampled=ADASYN().fit_sample(X, y)  #使用 ADASYN 方法过采样
print(sorted(Counter(y_resampled).items()))         #输出过采样后的样本维
                                                     度,并按类别排序
#欠采样算法-EasyEnsemble
from collections import Counter
from imblearn.ensemble import EasyEnsemble
ee=EasyEnsemble(random_state=0, n_subsets=10)
X_resampled, y_resampled=ee.fit_sample(X, y)      #使用 EasyEnsemble
```

```
                                                      方法欠采样
print(X_resampled.shape)                              #输出欠采样后的样本维度
print(sorted(Counter(y_resampled[0]).items()))  #输出欠采样后的样本维
                                                      度,并按类别排序
#阈值移动算法
from imblearn.under_sampling import InstanceHardnessThreshold
iht=InstanceHardnessThreshold(return_indices=True)
                    #初始化阈值移动算法处理类别不均衡样本,并输出样本的索引
X_res, y_res, idx_res=iht.fit_sample(X, y)            #调用阈值移动算法
```

第 3 章 基于 TensorFlow 用 Keras 做深度学习

流行的深度学习框架很多，如 TensorFlow、Caffe 和 Theano 等。但这些框架都是面向深度学习开发的专业人士的，用这些框架做深度学习需要非常了解底层逻辑，且需要编写大量的代码才能实现。这对非计算机编程相关专业科班出身的金融从业人员来说，是个很大的挑战，且短时间内很难弥补。本书介绍的深度学习库 Keras 是一个基于 TensorFlow 的 API，它能够使我们快速实现金融模型的思维，且易于编程实现。

3.1 Keras 简介

Keras 是由谷歌软件工程师 Francois Chollet 开发的、基于 TensorFlow 的深度学习库，具有较为直观的 API。目前，Keras 库已经成为 TensorFlow 的默认 API。Keras 库强调极简主义——只需要几行代码就能构建一个神经网络，其设计原则如下。

1. 用户友好

Keras 是神经网络的高层 API，它充分考虑用户的使用体验，在极大减少用户工作量的前提下，使用户非常方便地构建不同层次深度的定制化神经网络模型。

2. 模块化

使用 Keras 构建深度学习模型时，可将该模型理解为一个多层的序列或数据的运算图，这些完全可配置的模块可以用最少的代价自由组合在一起。具体而言，网络层、损失函数、优化器、初始化策略、激活函数、正则化方法等都是独立的模块，可以使用它们来构建自己的模型。

3. 易扩展性

添加新模块（或网络层、神经元等）非常容易，只需要仿照现有的模块编写新的类或函数即可。创建新模块的便利性使得 Keras 更适合于先进的研究工作。

为了说明使用 Keras 进行深度学习模型开发的便利性，此处以一个简单的

示例说明如何使用 Keras 库做深度学习模型开发。该示例以上证 50 成分股的收盘价来训练模型,并使用训练得到的深度学习模型计算上证 50 指数的预测收盘价,最后绘图表示上证 50 实际收盘价和预测收盘价的曲线,示例代码如下所示。

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation,Dropout
from sklearn.preprocessing import Normalizer, StandardScaler
import matplotlib.pyplot as plt
from WindPy import w
#####获取数据并生成数据集######
#获取上证 50 成分股,并获取这些股票从 2010 年 1 月 1 日到
#2017 年 12 月 31 日所有交易日的收盘价和在此期间上证 50
#的收盘价
#目标:用上证 50 成分股的收盘价预测上证 50 指数的收盘价
w.start()#本段代码需要安装 Wind 资讯金融终端和 WindPy 插件才能运行
wsetdata=w.wset("sectorconstituent","date=2018-02-06;sectorid=
1000000087000000")      #获取上证 50 成分股的证券代码
code_list=wsetdata.Data[1]
code_list.append('000016.SH')
codes=code_list[0]
for code in code_list[1:]:
    codes=codes + ',' + code
w_data=w.wsd(codes, "close", "2010-01-01", "2017-12-31", "Fill=
Previous")              #获取所有成分股和上证 50 指数的收盘价
close_data=pd.DataFrame(w_data.Data,index=w_data.Codes,columns=w_
data.Times)
close_data=close_data.T
all_data=close_data.dropna(axis=1,how='any')     #剔除存在 nan 的列
all_data.index.name="Date"
all_data.to_csv("sz50_all_data.csv")     #将获取的数据集写入 csv 文件
#####获取数据并生成数据集######
df=pd.read_csv('sz50_all_data.csv', index_col='Date')
df.index=pd.to_datetime(df.index)
df=df.resample('W-MON').last()
df=df.dropna(axis=0,how='any')           #剔除存在 nan 的行
stox=list(df.columns)
stox.remove('000016.SH')
```

```
ind='000016.SH'
df['ret']=df['000016.SH'].pct_change().fillna(0.1) #计算上证 50 指数
当日收盘价相对前一收盘价的涨跌百分比,第一个值为 0.1
df.loc[:, 'new_ret']=df.apply(lambda r: 0.1 if r['ret'] <  -0.08 else
r['ret'], axis=1)
df['new_index']=df.loc[df.index[0], '000016.SH']
for i in range(len(df.index)):
    if i >  0:
        df.loc[df.index[i], 'new_index']=df.loc[df.index[i-1],
'new_index']* (1.0+df.loc[df.index[i], 'new_ret'])
#将所有数据,按照列标准化
en=StandardScaler()
df['000016.SH']=en.fit_transform(np.array(df['000016.SH']).reshape
(-1,1))
for s in stox:
    en1=StandardScaler()
    df[s]=en1.fit_transform(np.array(df[s]).reshape(-1,1))
df['new_index']=en.transform(np.array(df['new_index']).reshape
(-1,1))
train=df[df.index <  pd.to_datetime('29-12-2017')]
def create_dataset_index(df):
    dataX=[]
    dataY=[]
    for i in df.index:
        x=df.loc[i, stox].as_matrix()
        dataX.append(x)
        y=df.loc[i, '000016.SH']
        dataY.append(y)
    return np.array(dataX), np.array(dataY)
#训练样本中,train_x 为所有成分股,train_y 为上证 50 指数
train_x, train_y=create_dataset_index(train)
#构建四层顺序神经网络模型
model=Sequential()
model.add(Dense(output_dim=20, input_dim=36))   #第一层,36 个输入、
                                                  20 个输出节点
model.add(Activation('tanh'))
model.add(Dropout(0.1))
model.add(Dense(output_dim=10, input_dim=20))   #第二层,20 个输入、
                                                  10 个输出节点
model.add(Activation('tanh'))
model.add(Dropout(0.1))
model.add(Dense(output_dim=5, input_dim=10))    #第三层,10 个输入、
```

```
                                                    5个输出节点
model.add(Activation('tanh'))
model.add(Dropout(0.1))
model.add(Dense(output_dim=1, input_dim=5))      #第四层,5个输入、
                                                    1个输出节点
model.compile(loss='mean_squared_error', optimizer='rmsprop')#编译
模型
model.fit(train_x, train_y, nb_epoch=2000, batch_size=50, verbose=
1)#拟合模型
#使用模型计算上证50指数收盘价
df['pred']=0.0
df=df[df.index < pd.to_datetime('29-12-2017')]
for i in df.index:
    x=df.loc[i, stox].as_matrix()
    df.loc[i, 'pred']=model.predict(x.reshape(1,36))[0,0] #使用训练
    出的模型,计算上证50指数收盘价
#将实际上证50指数和按预测模型计算的指数,做逆运算、变为实际收盘价
df['pred']=en.inverse_transform(df['pred'])
df['000016.SH']=en.inverse_transform(df['000016.SH'])
df.to_hdf('DeepLearning-sz50.h5', 'Deep_Portfolio')
plt.plot(df['pred'], 'r', label='pred')
plt.plot(df['000016.SH'], 'b', label='index')
plt.legend()
plt.show()     #输出实际指数与预测指数比较图
```

上证50实际指数用蓝色表示，预测指数用红色表示，如图3.1所示。可见，用深度学习模型得到的预测结果跟实际结果非常接近，即模型的拟合度较好。

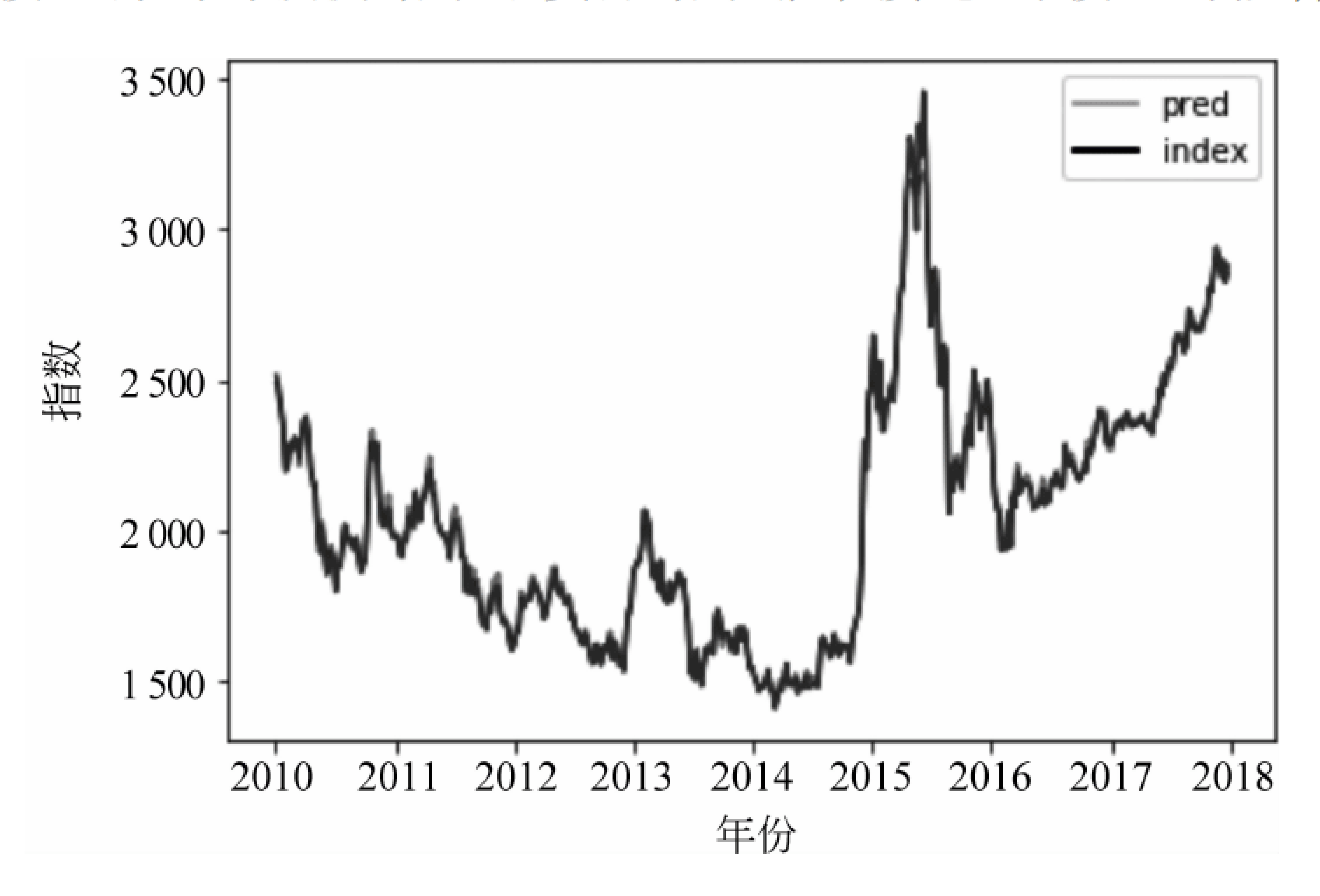


图3.1　实际指数与预测指数比较

3.2 Keras 安装与配置

由于深度学习的框架基本都是基于 Linux 平台开发的，因此在 Windows 平台下安装深度学习库一般比较困难。此处，我们介绍一种在 Windows 平台下安装虚拟机进行深度学习开发的方法。

首先，安装虚拟机软件 VirtualBox。访问 VirtualBox 首页(https://www.virtualbox.org/)，并下载、安装 VirtualBox，如图 3.2 所示。



图 3.2 VirtualBox 虚拟机首页

安装 VirtualBox 时，按照默认选项直接安装完成即可。

接下来，需要下载 Ubuntu Linux 操作系统的镜像文件(iso)。访问 Ubuntu 桌面版操作的首页(https://www.ubuntu.com/download/desktop)，下载 Ubuntu 16.04.3 LTS，如图 3.3 所示。下载得到的 Ubuntu Linux 系统的文件名为 ubuntu-16.04.3-desktop-amd64.iso。

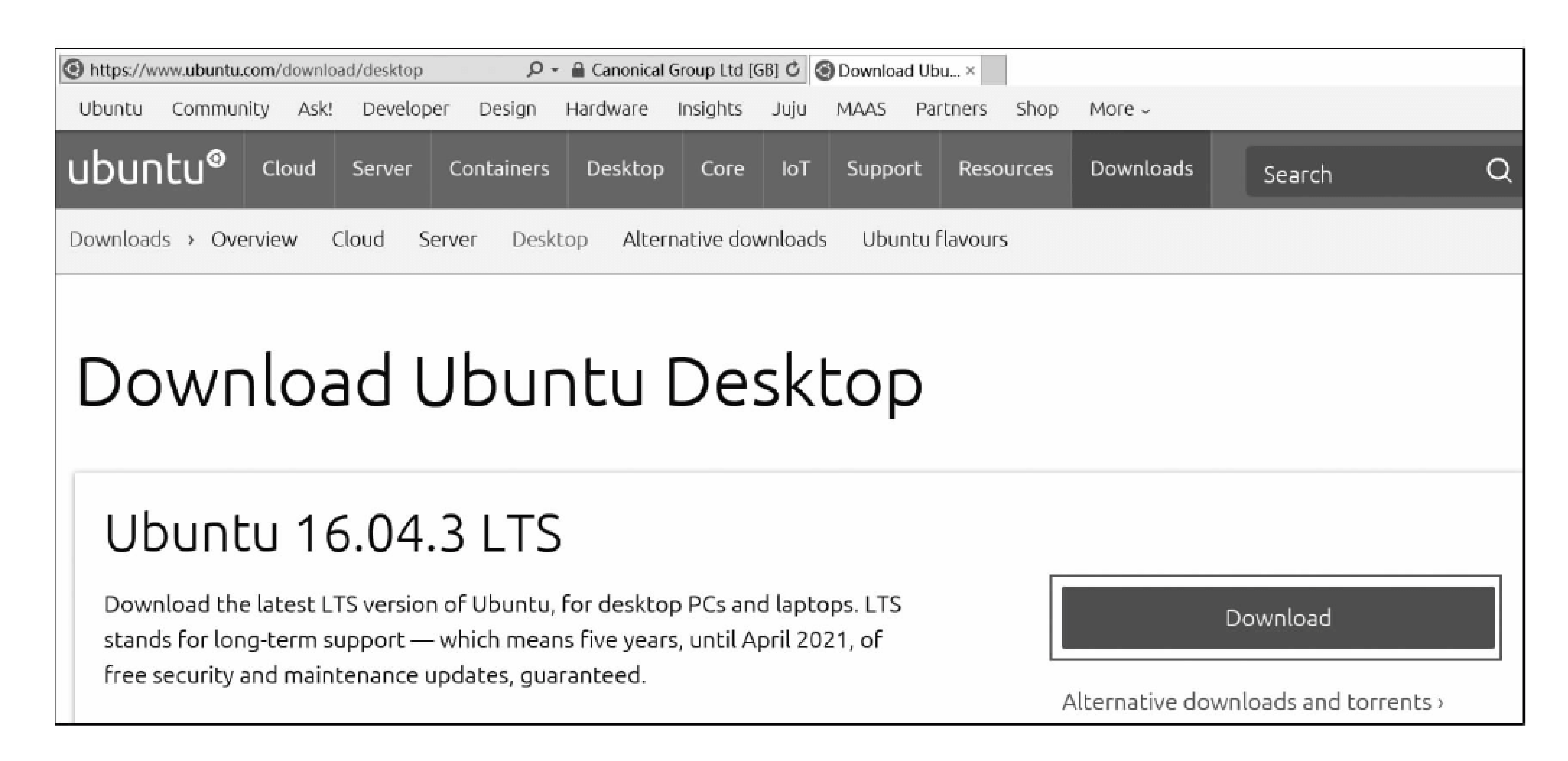


图 3.3 Ubuntu 16.04.3 Desktop

启动已经安装好的 VirtualBox。右击 VirtualBox 图标,并选择"以管理员身份运行"。单击"新建"按钮,会显示图 3.4 所示的启动画面。此时,我们需要填写虚拟机的名称、类型、版本等信息。此处,我们将虚拟机命名为 DeepLearning,类型为 Linux,版本为 Ubuntu(64-bit)。

为新建的虚拟机分配内存。如果读者处理的数据量较大,建议至少分配 4 GB(4 096 MB)内存,本书以分配 2 GB(2 048 MB)内存为例,如图 3.5 所示。

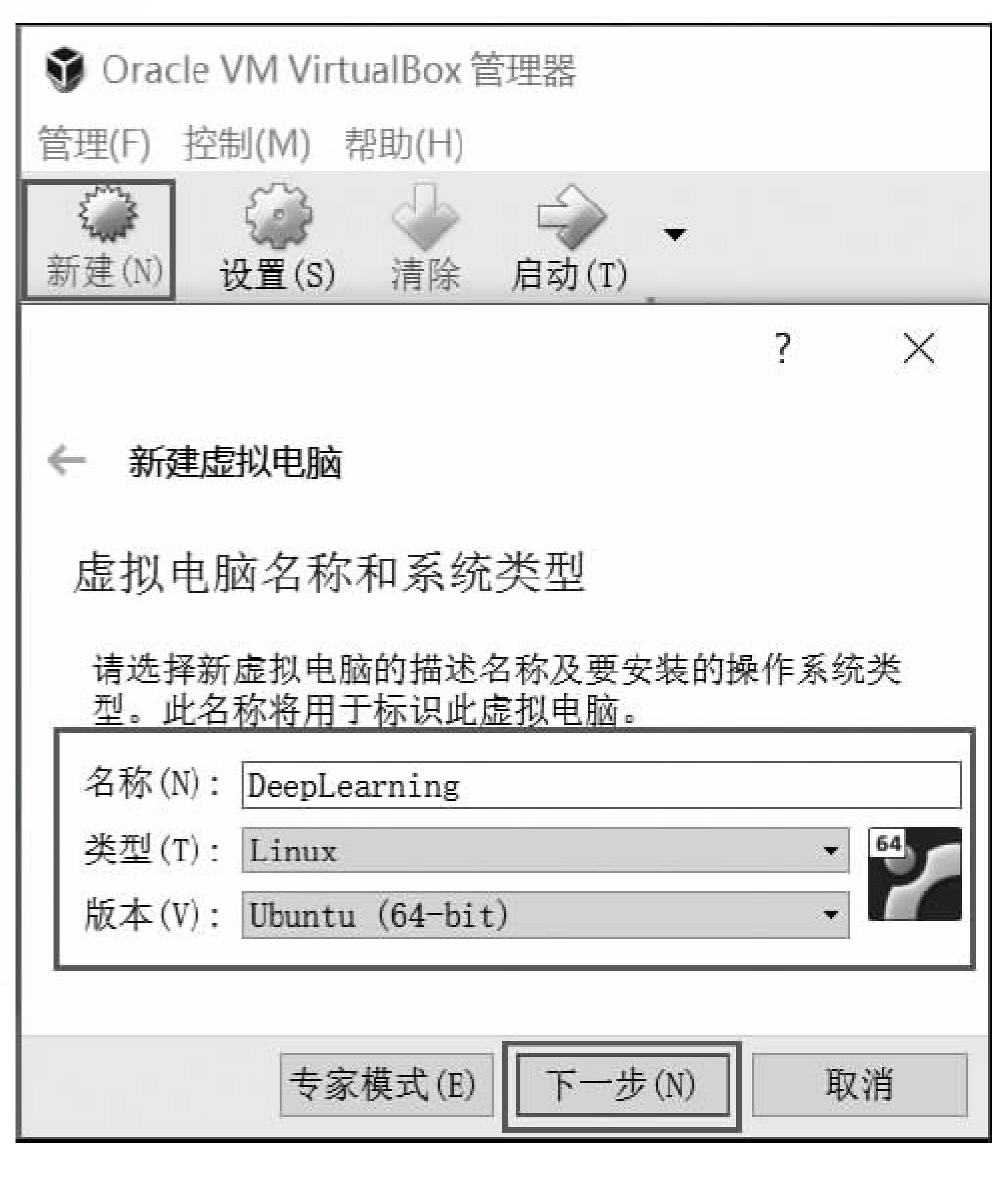


图 3.4　启动 VirtualBox 并安装虚拟机



图 3.5　设置虚拟机内存大小

为虚拟机创建虚拟硬盘。选择"现在创建虚拟硬盘",如图 3.6 所示。

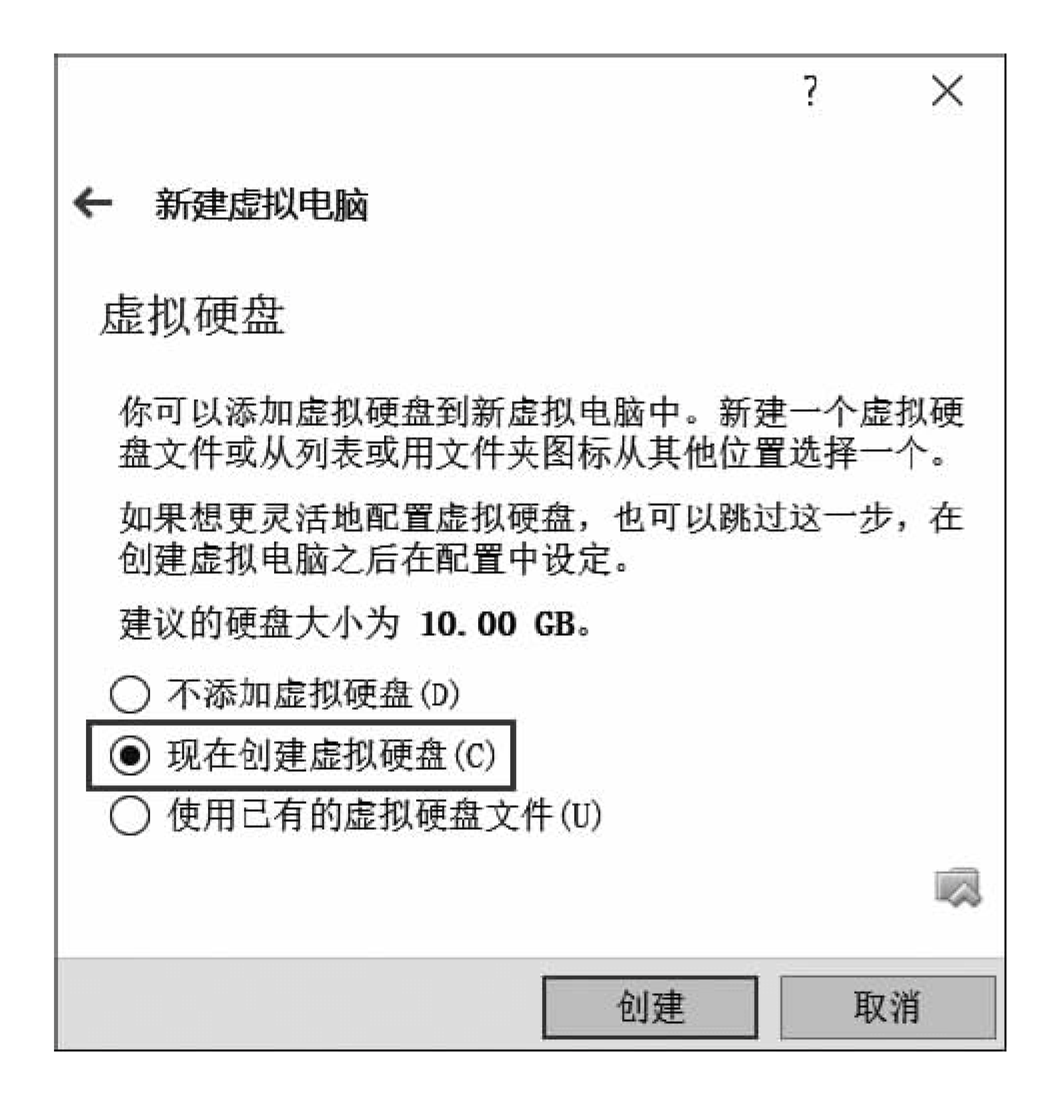


图 3.6　创建虚拟硬盘

为虚拟硬盘选择文件类型。建议选择默认文件类型 VDI,如图 3.7 所示。

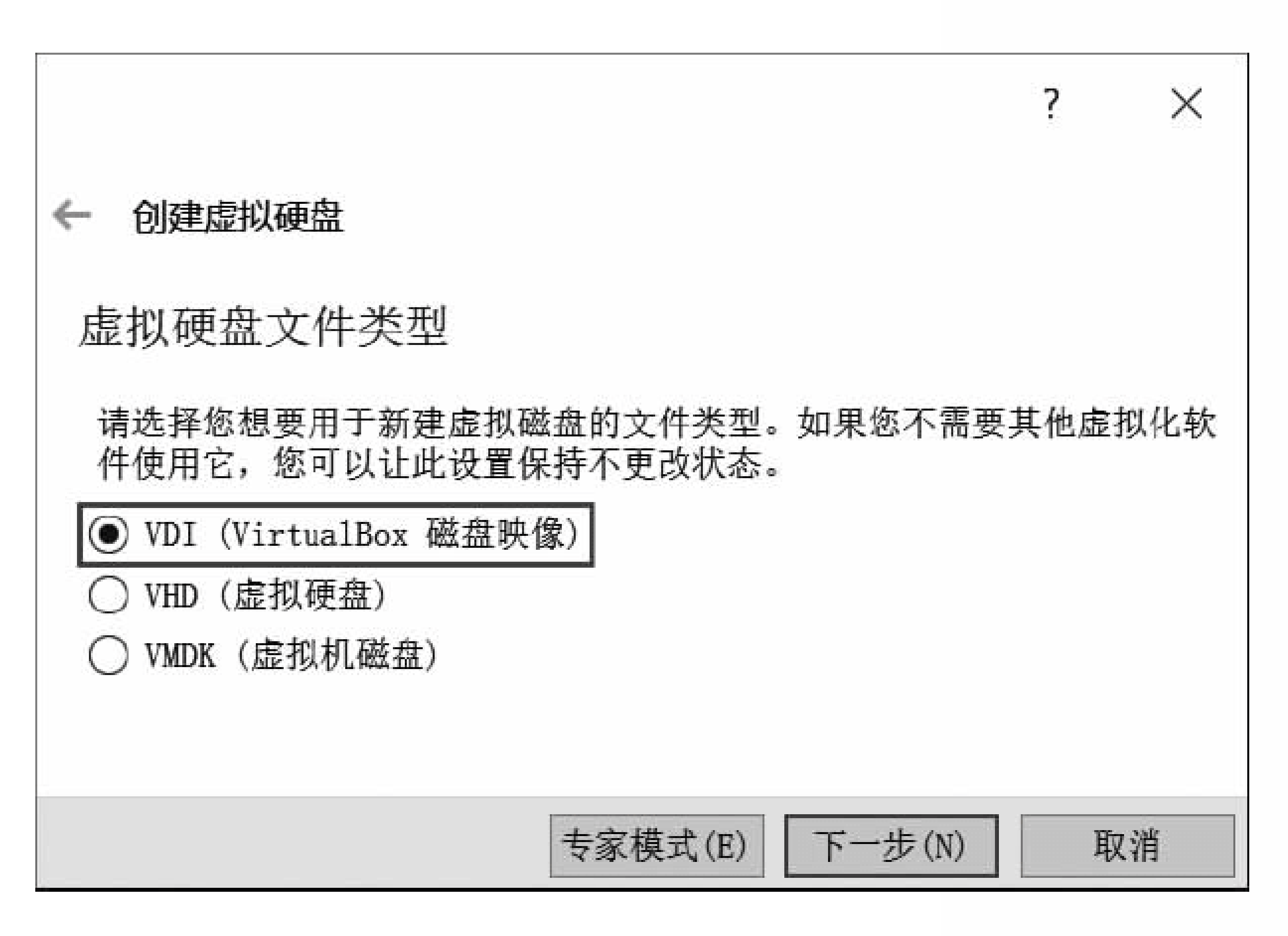


图 3.7　选择虚拟硬盘文件类型

选择虚拟硬盘在物理硬盘上的存储方式。此处，选择固定大小，以便提升虚拟机的运行速度，如图 3.8 所示。

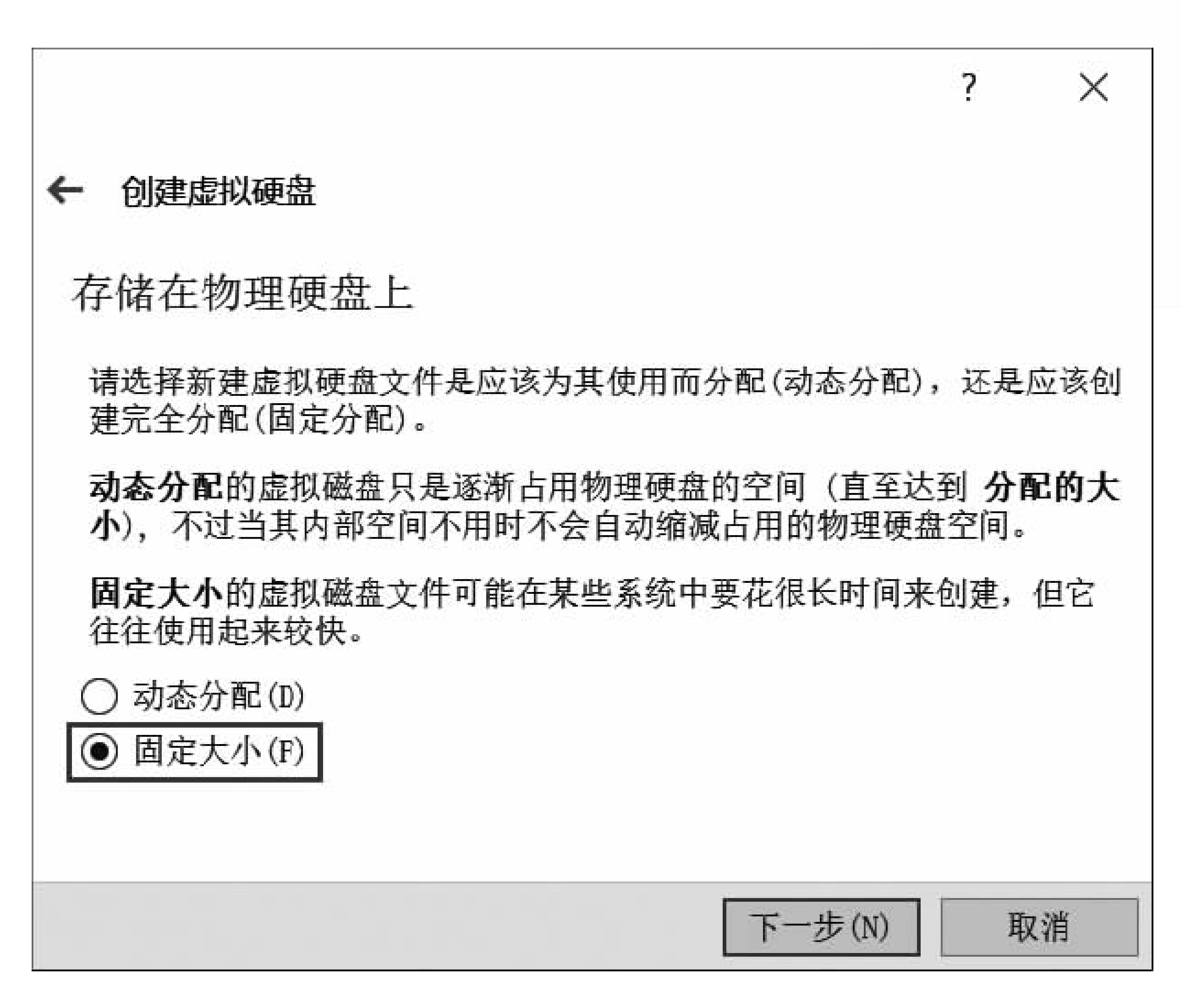


图 3.8　虚拟磁盘分配方式

选择虚拟硬盘在 Windows 上的存储位置。此时，需要首先在 Windows 系统上找到剩余存储空间较大的硬盘(本书是 F 盘)，并新建一个文件夹(本书命名为 Virtual Box)。虚拟硬盘的大小建议至少选择 10 GB，以 20 GB 为佳，如图 3.9 所示。

虚拟机创建完成，接下来安装 Ubuntu Linux 操作系统。选择“DeepLearning”并单击“启动”按钮，如图 3.10 所示。

选择 Linux 系统盘。找到已经下载的 Linux 系统盘的 ISO 镜像文件，并选择“ubuntu-16.04.3-desktop-amd64.iso”，如图 3.11 所示。

操作系统语言选择“English”，然后单击“Install Ubuntu”按钮，如图 3.12 所示。

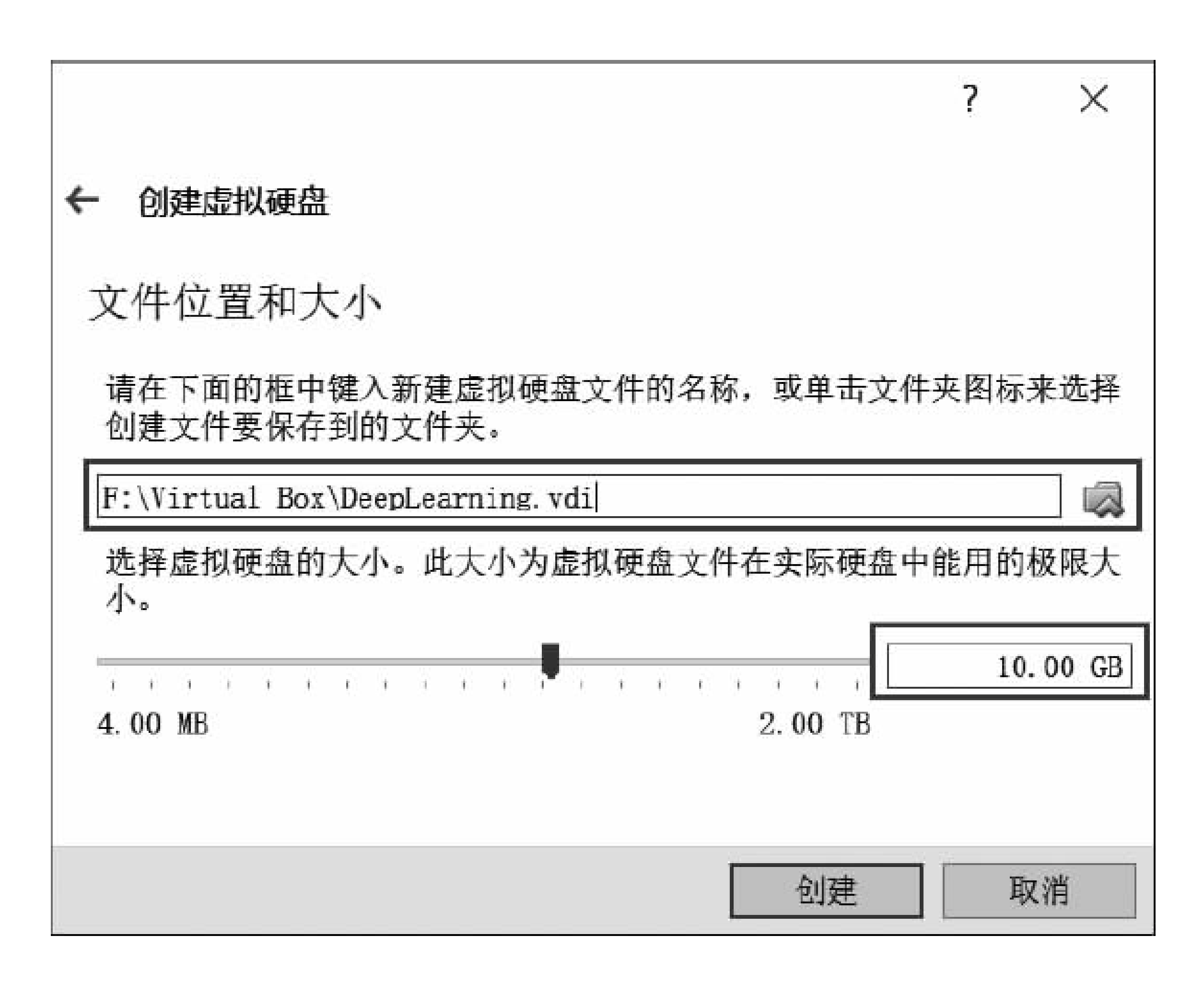


图 3.9　虚拟硬盘位置和大小

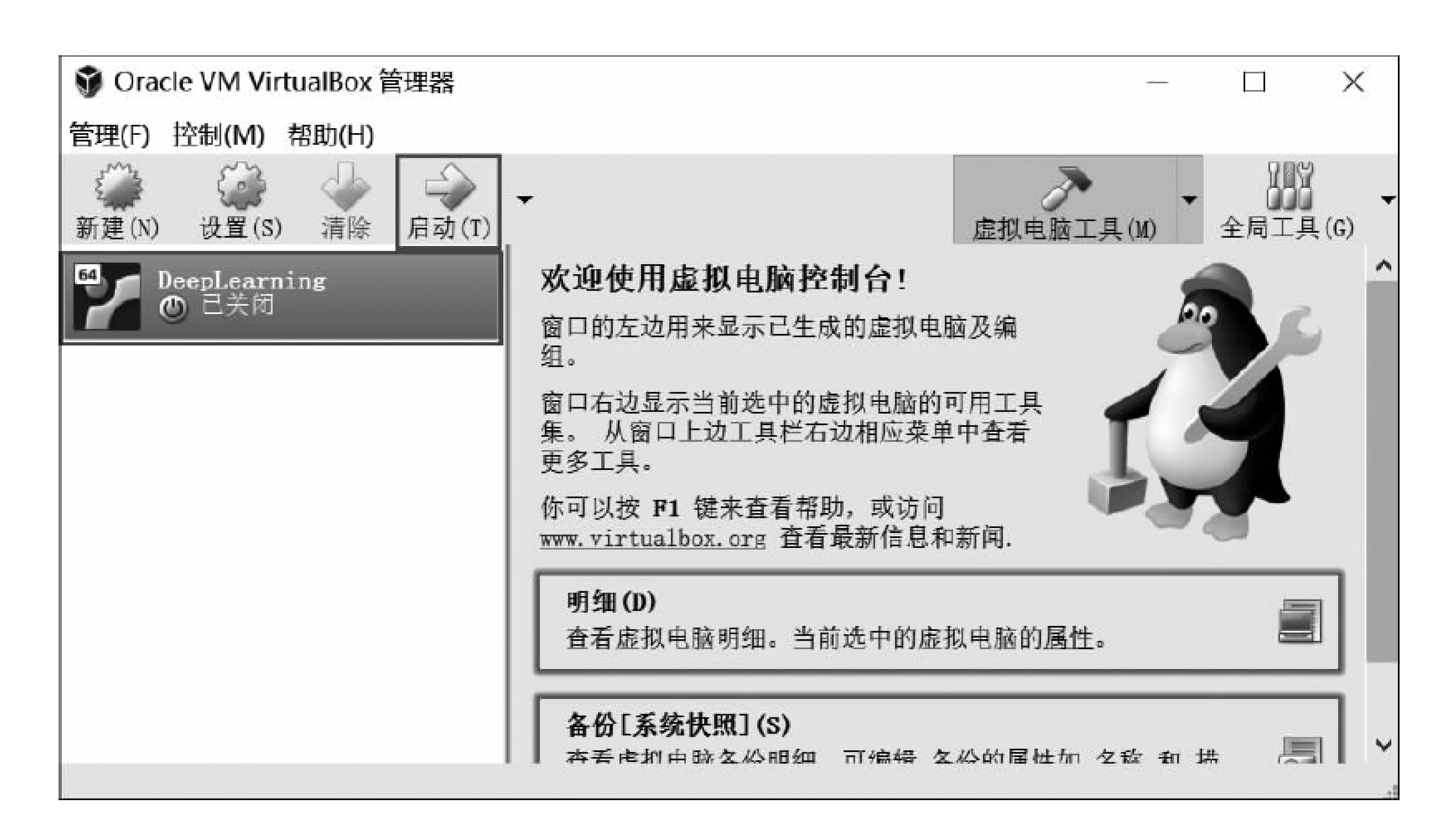


图 3.10　启动新创建的虚拟机

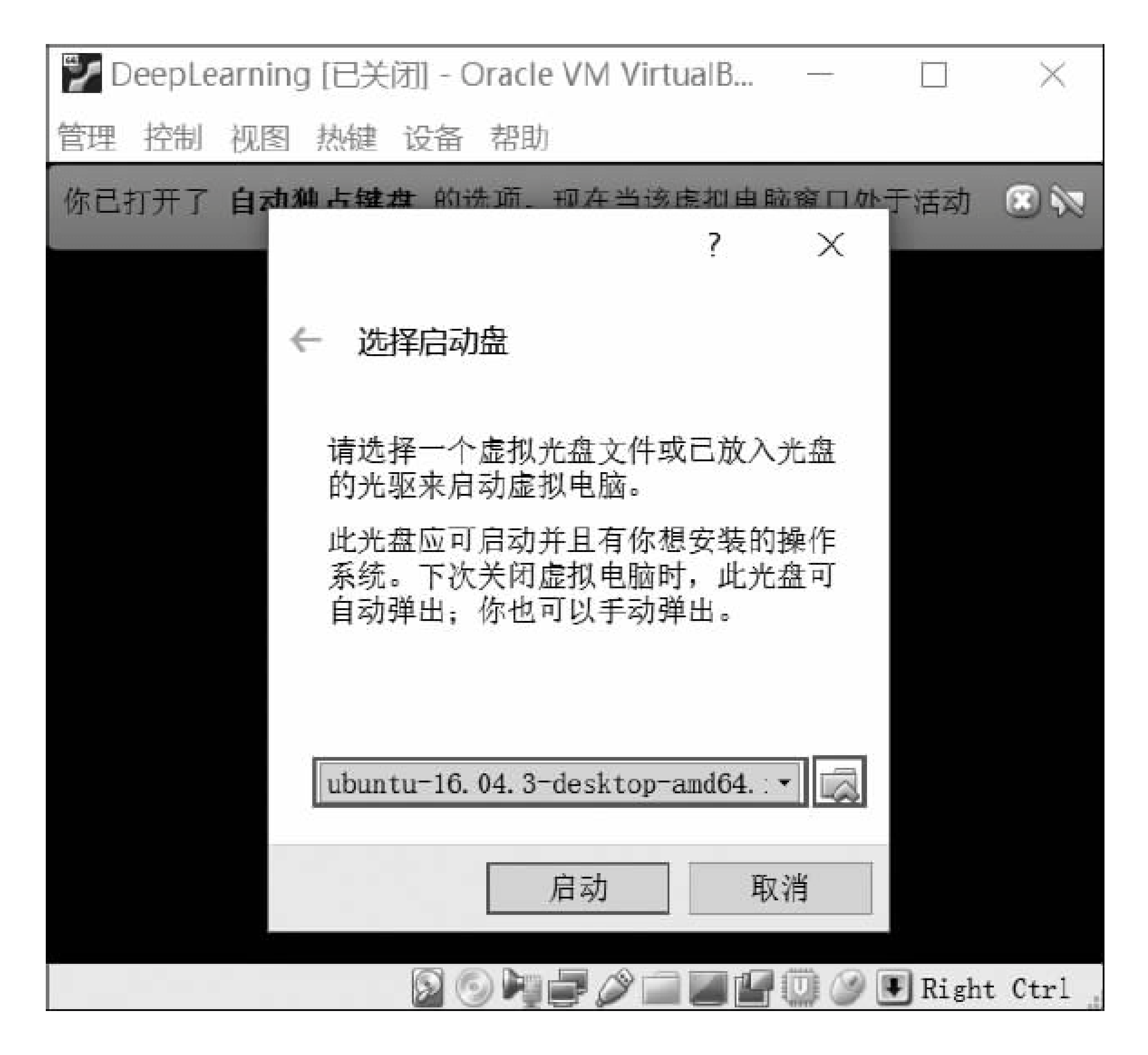


图 3.11　选择下载的 Ubuntu Linux 系统盘



图 3.12 安装 Ubuntu

为提升安装速度，此处建议不选择安装操作系统的附加项，如图 3.13 所示。

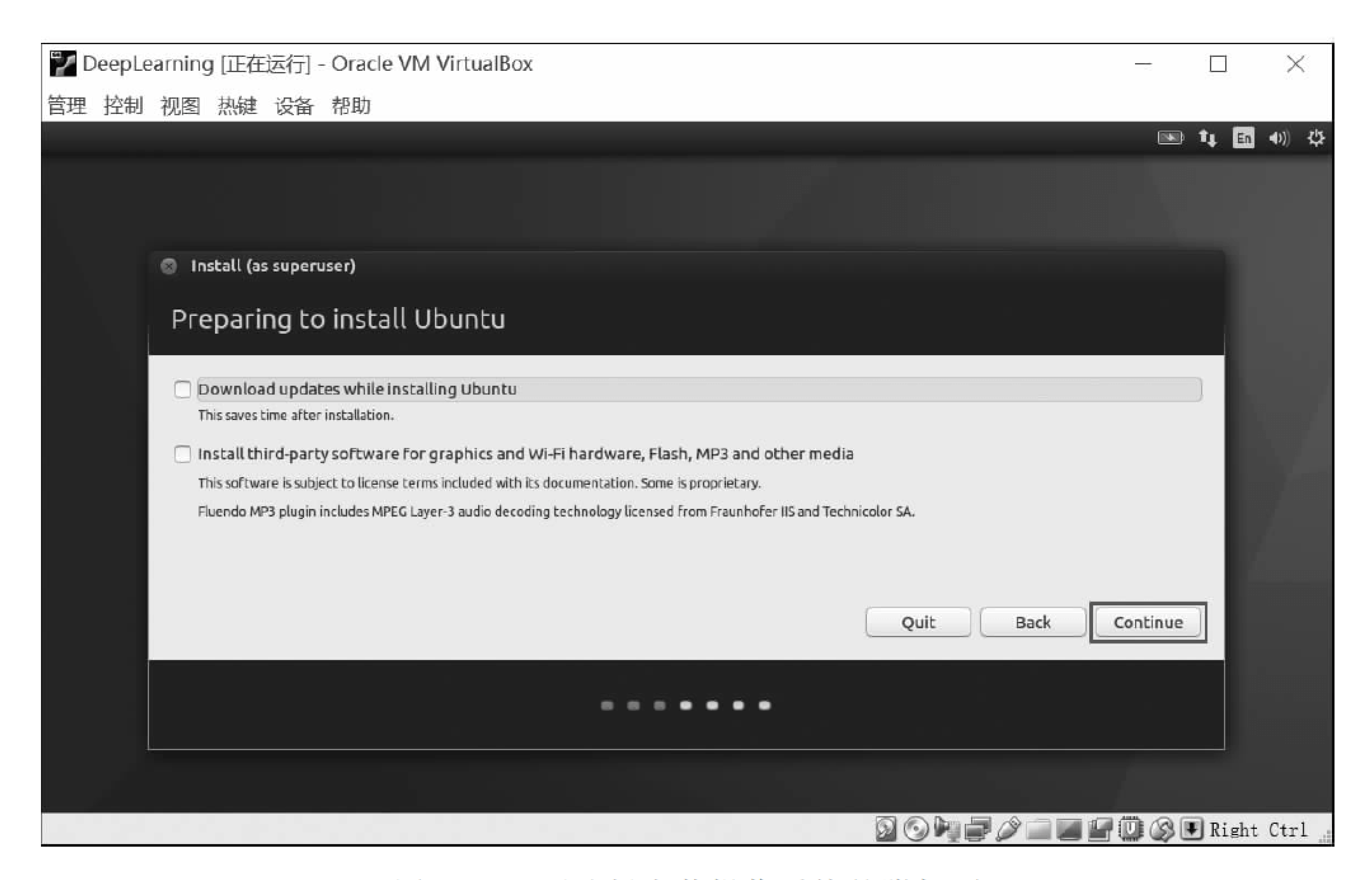


图 3.13 不选择安装操作系统的附加项

选择安装选项。建议选择格式化虚拟硬盘并安装操作系统（Erase disk and install Ubuntu），如图 3.14 所示。

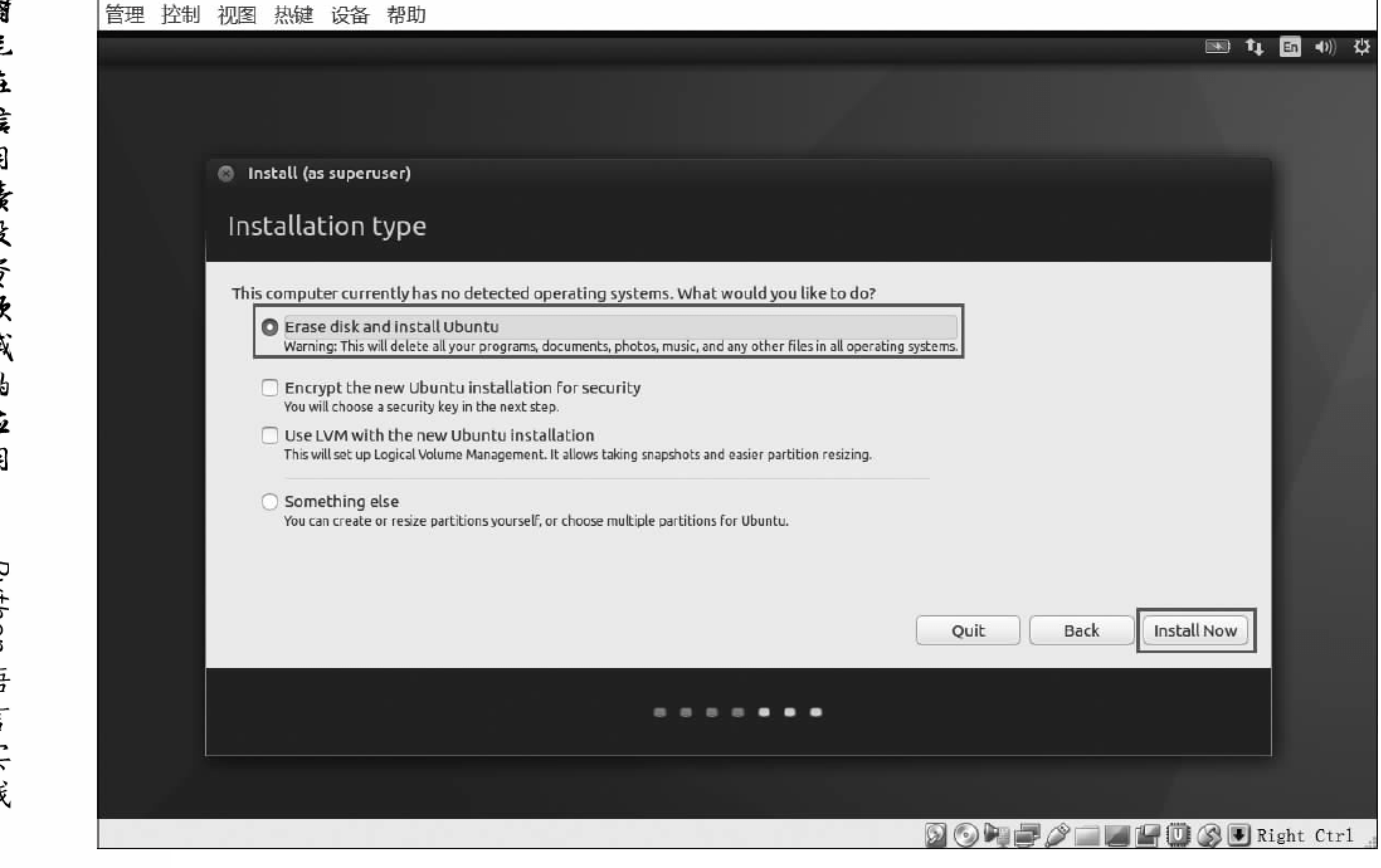


图 3.14　选择安装类型

单击“Continue”，安装 Ubuntu 操作系统，如图 3.15 所示。

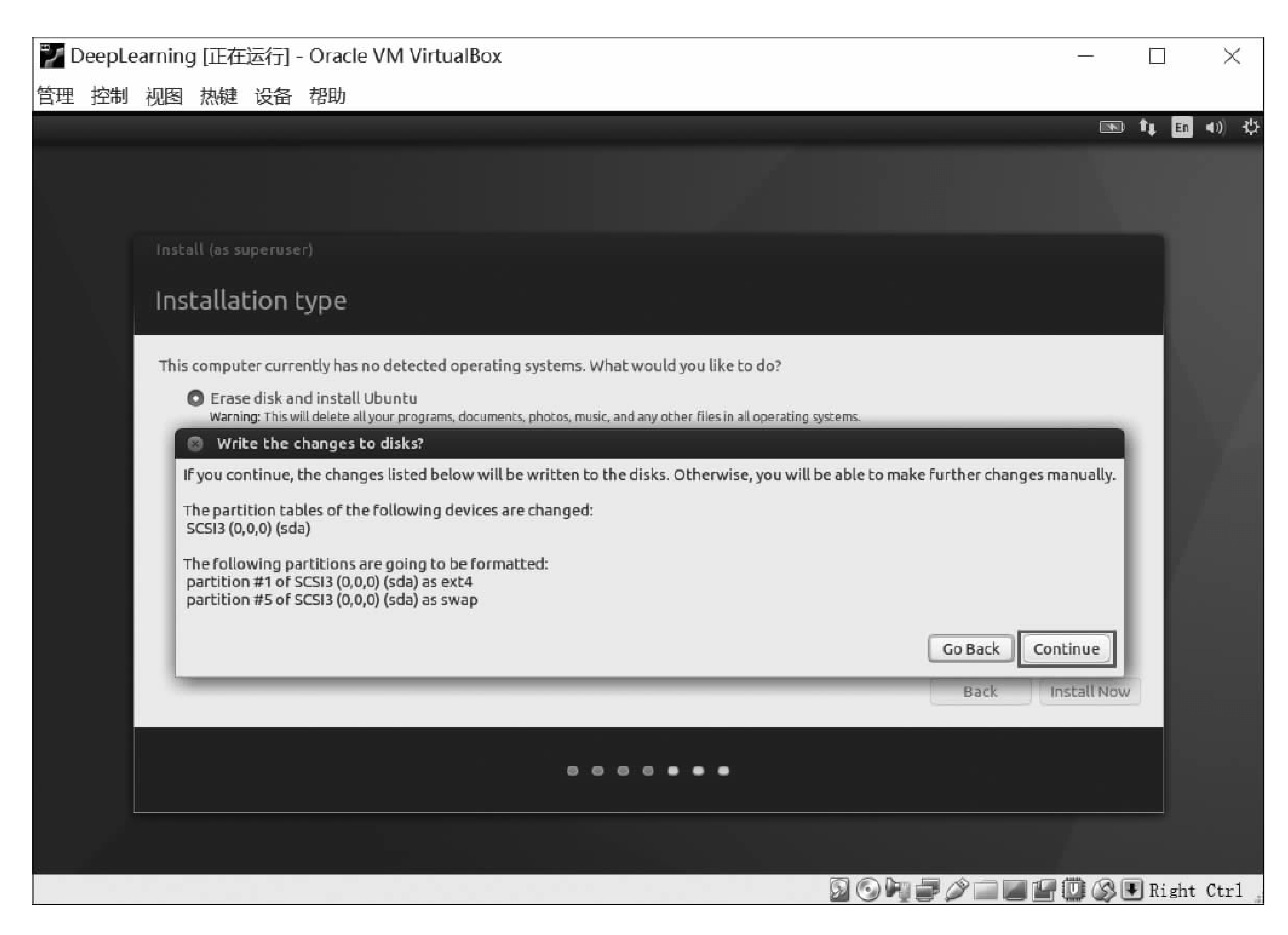


图 3.15　确认安装类型

选择时区。中国的时区在 Ubuntu 中只能选择“Shanghai”,如图 3.16 所示。



图 3.16　选择时区

选择键盘类型。默认都是选择 English(US),如图 3.17 所示。

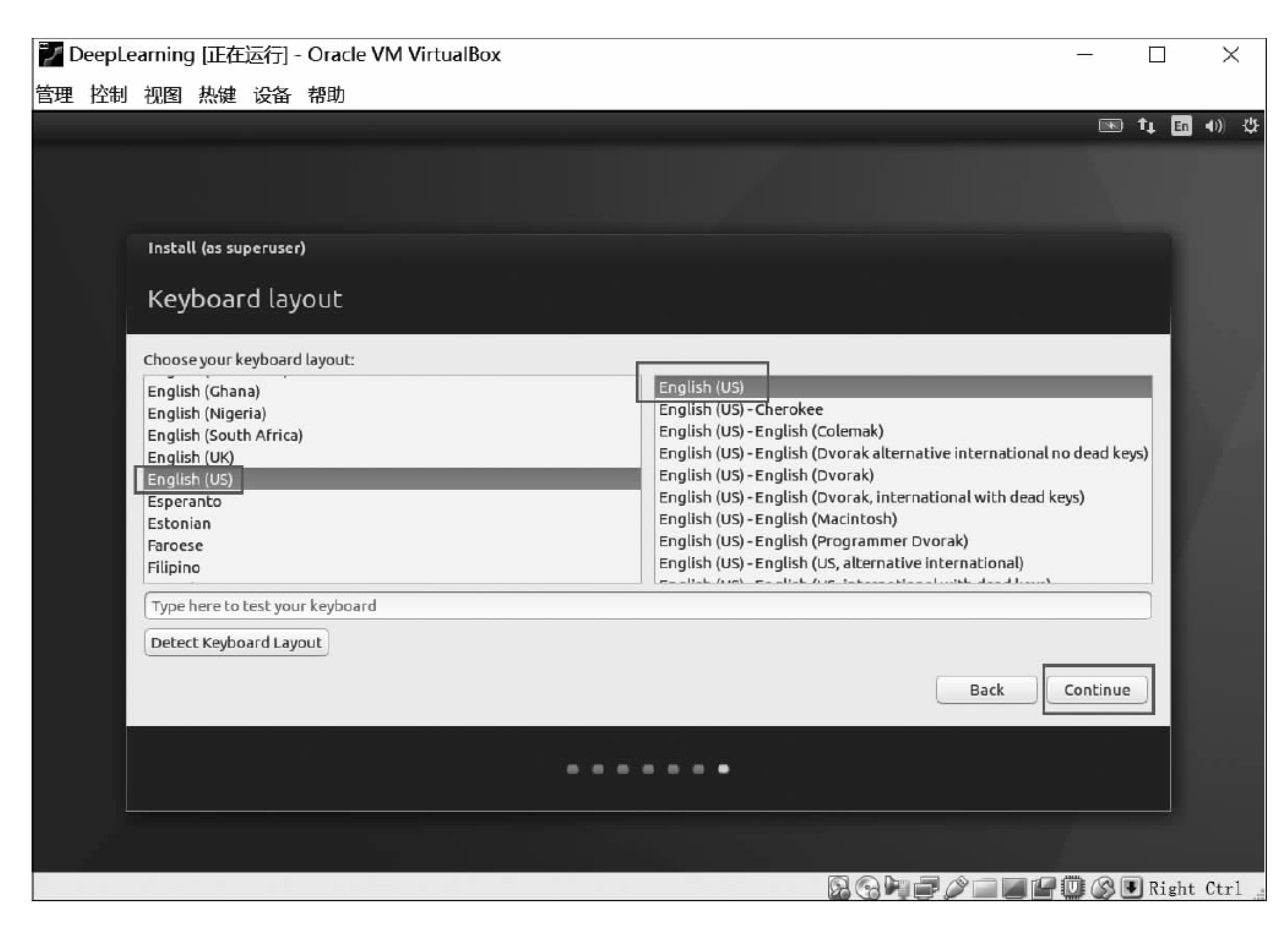


图 3.17　选择键盘类型

填写用户名和密码，建议名称不要太长，如图 3.18 所示。

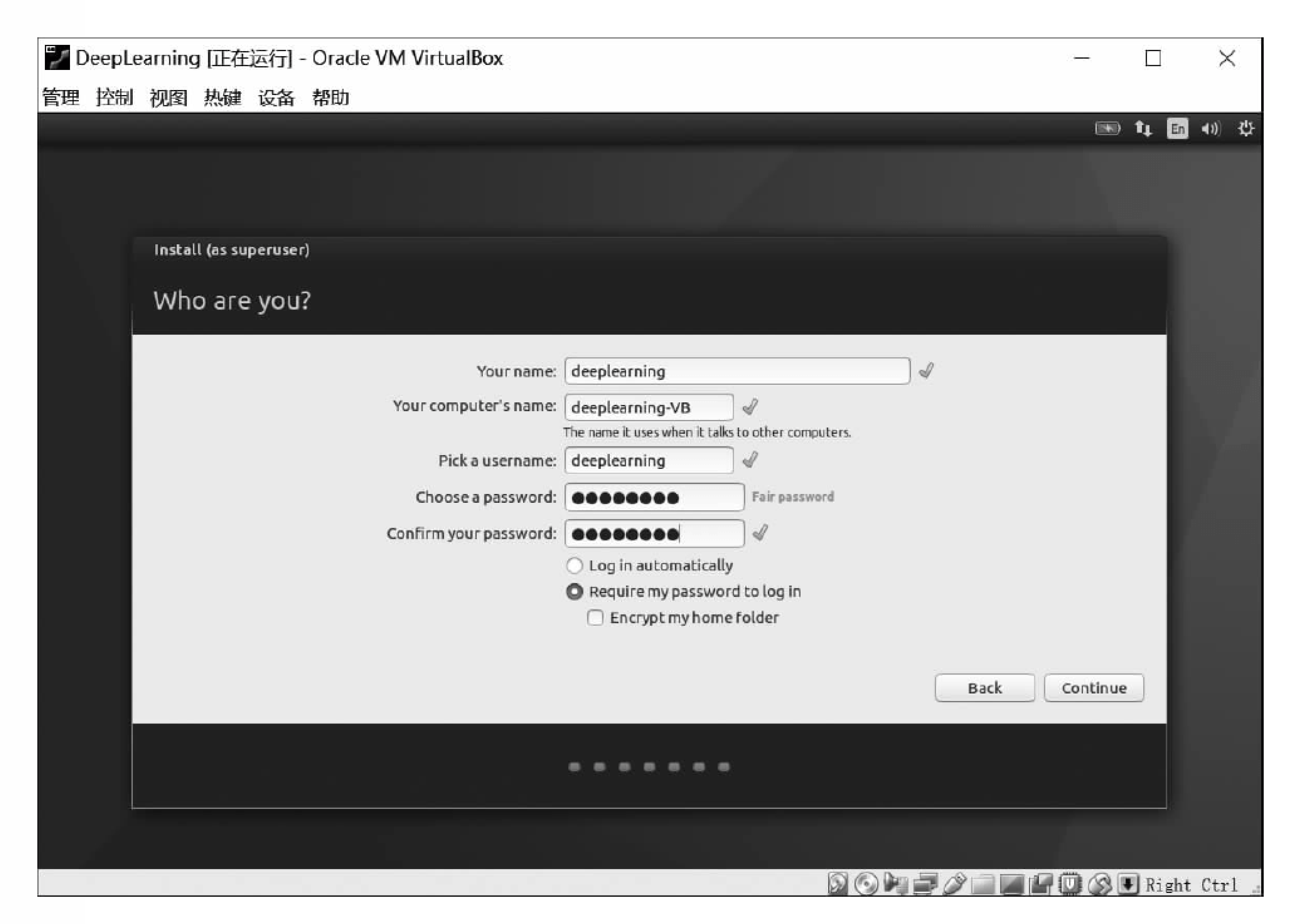


图 3.18 填写用户名和密码

接下来开始安装 Ubuntu Linux 操作系统，安装完成后，需要重新启动操作系统，如图 3.19 所示。

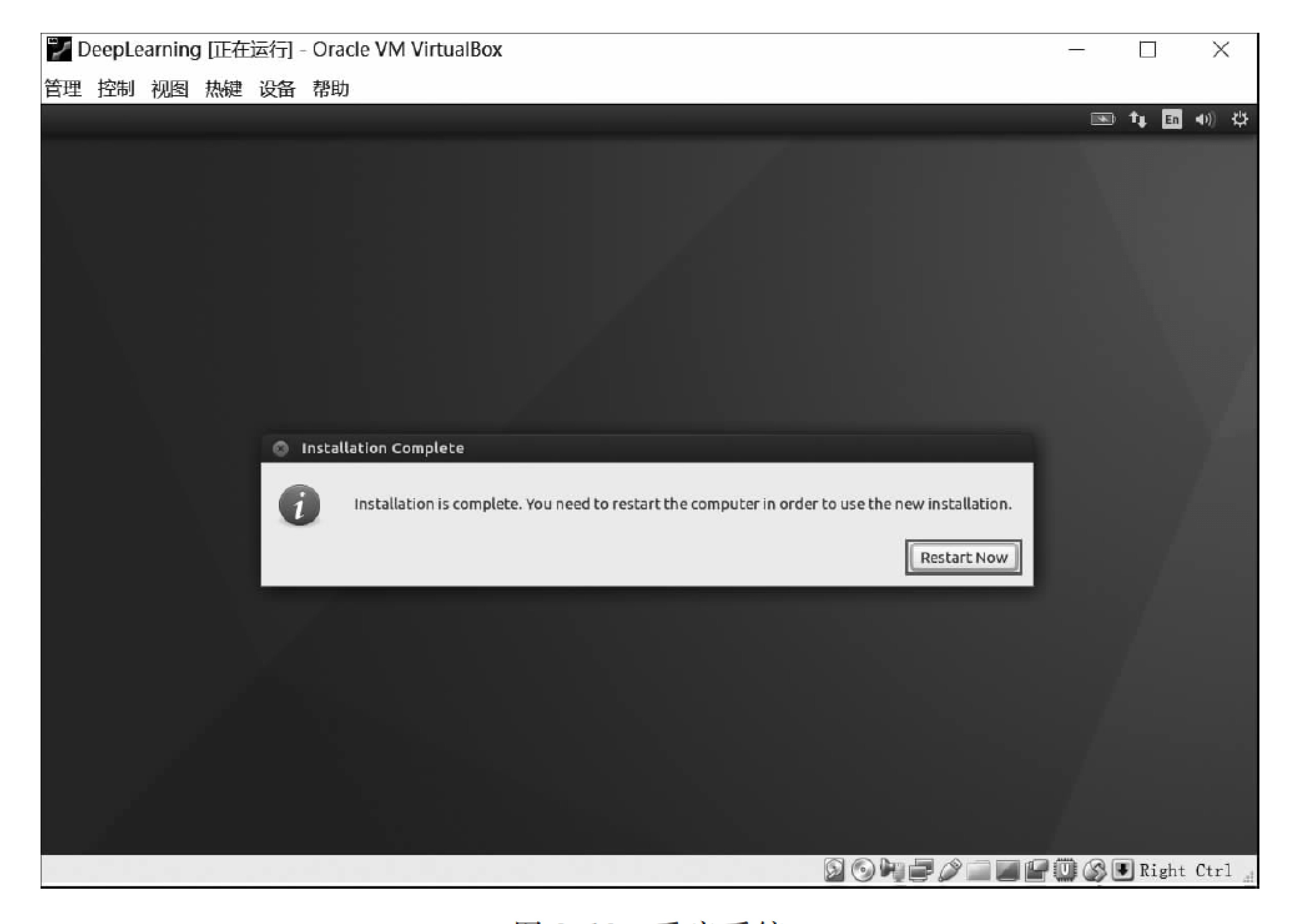


图 3.19 重启系统

按照设置的用户名和密码，登录操作系统，如图3.20所示。

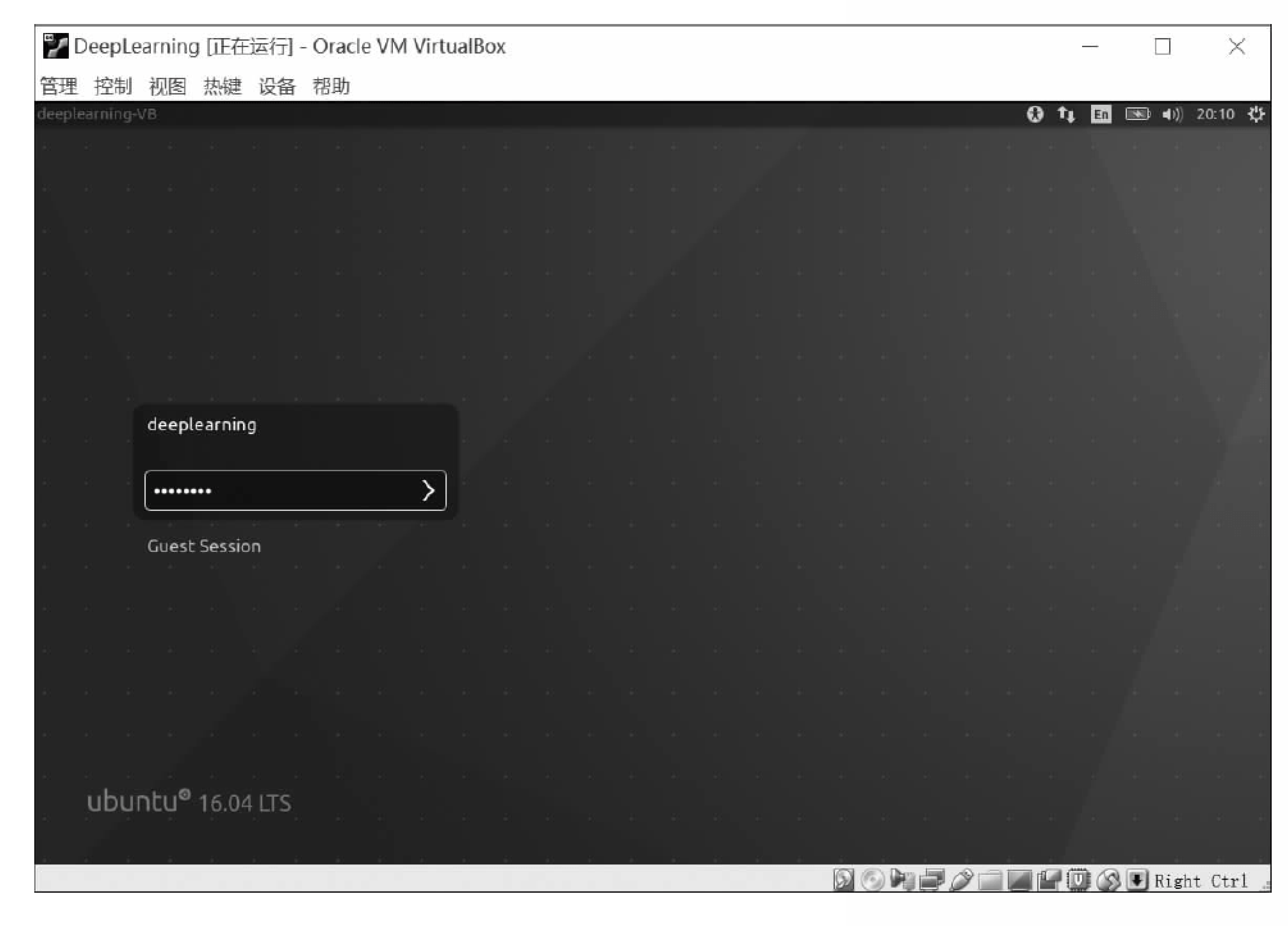


图3.20　登录Ubuntu

在Ubuntu Linux系统桌面，右击，在弹出的对话框中选择“Open Terminal”，打开Linux命令行，如图3.21所示。

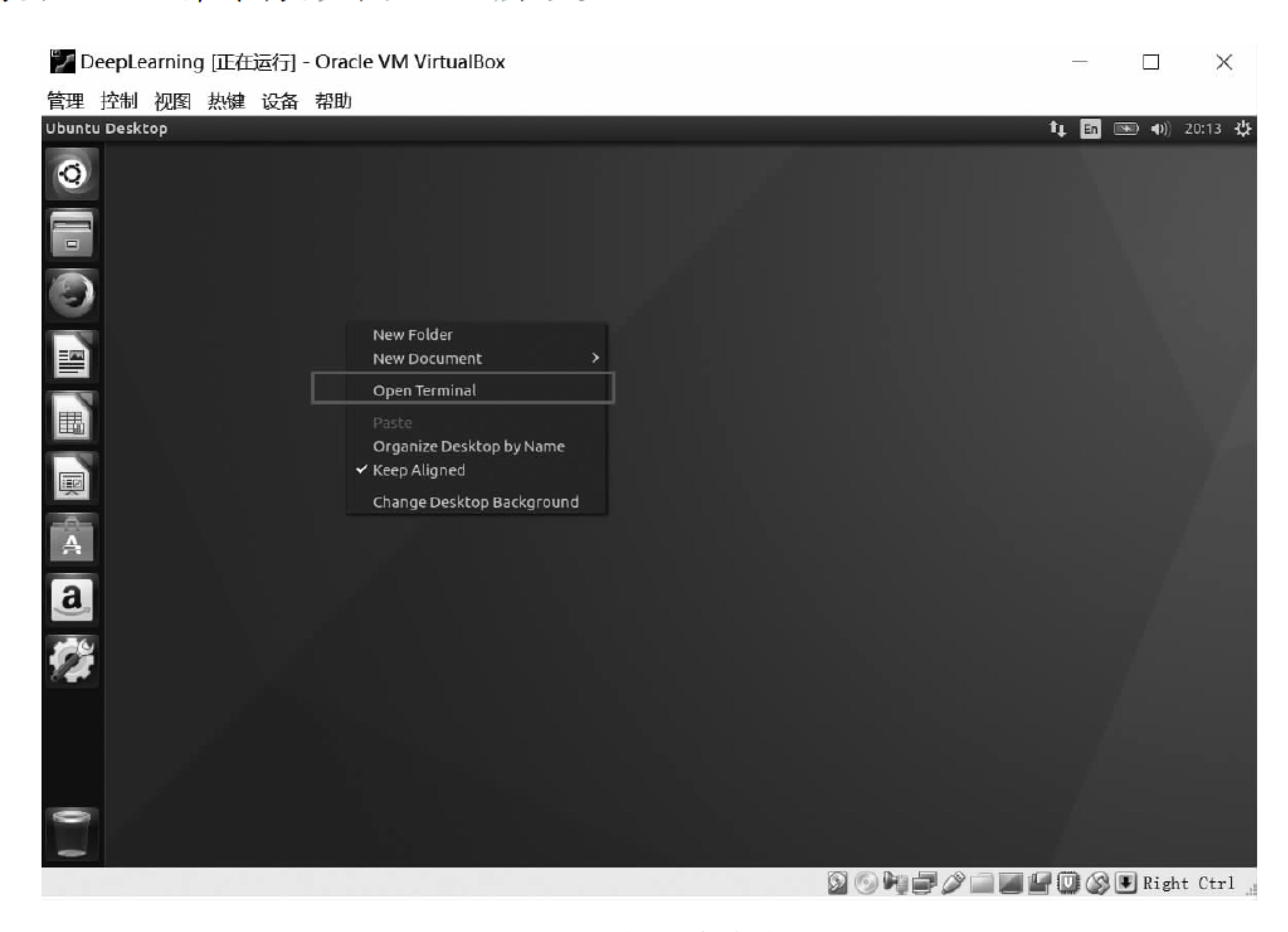


图3.21　打开命令窗口

为避免每次用鼠标右键启动命令行窗口，可固定到左侧启动栏。在左侧启动栏选择 Terminal 终端的图标，并右击，选择“Lock to Launcher”，即可将命令行窗口固定到启动栏，下次再启动命令行窗口时，只需要单击启动栏的命令行窗口图标即可，如图 3.22 所示。

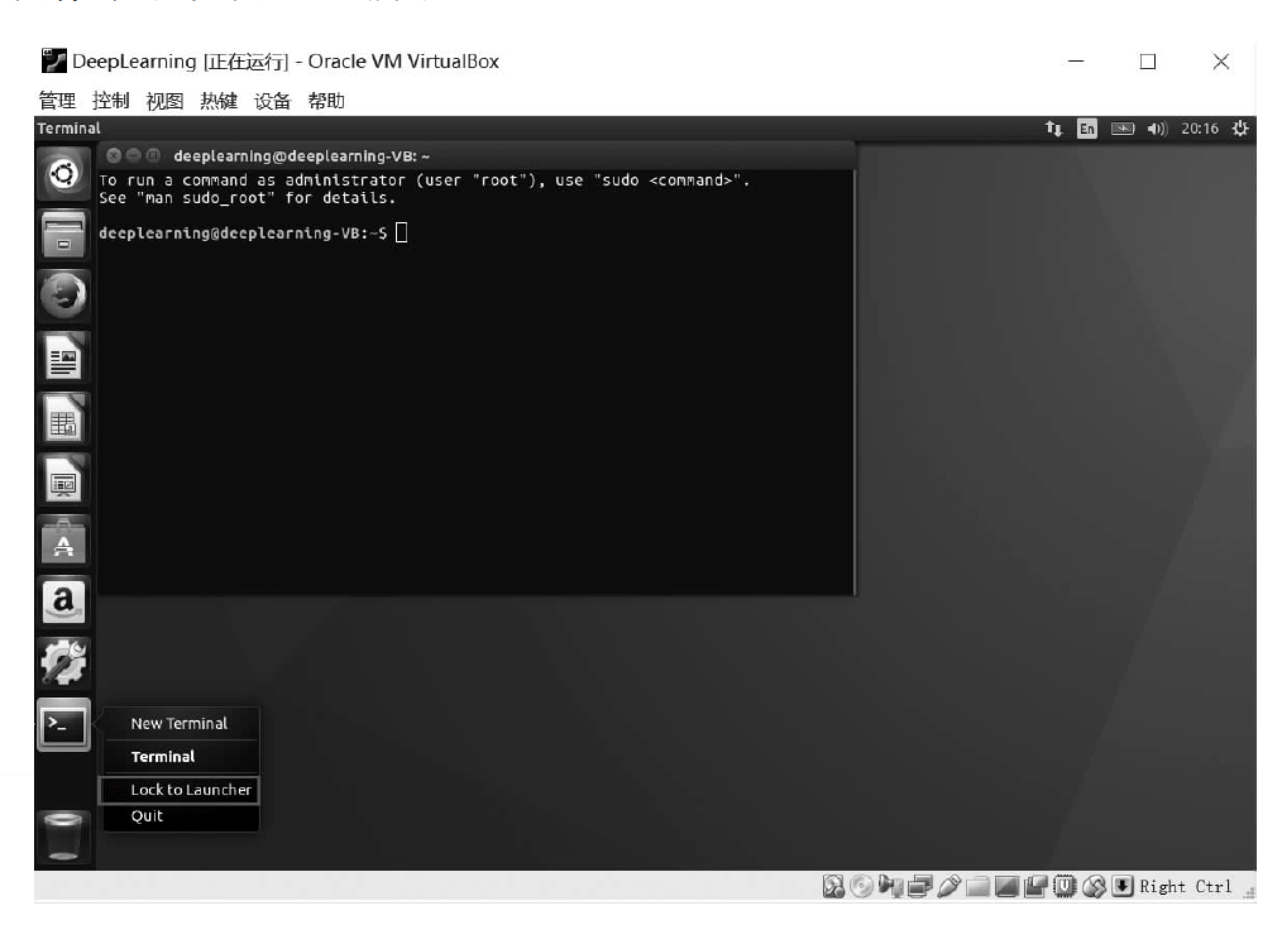


图 3.22　固定命令窗口

下载 Linux 版本的 Python 集成开发环境 Anaconda。首先，访问 Anaconda 首页(https://www.anaconda.com/download/#linux)，下载 Python 3.6 版本的 Anaconda，如图 3.23 所示。

下载完成的 Anaconda 在 Downloads 文件夹下，如图 3.24 所示。

打开命令行窗口(Terminal)，用 ls 命令查看当前目录下的所有文件和文件夹，并使用 cd 命令进入 Downloads 文件夹。用 bash 命令安装下载的 Anaconda，如图 3.25 所示。在开始安装 Anaconda 之前，需要多次按回车键，以浏览 License。

输入 yes 并按回车，再次按回车后开始安装 Anaconda，如图 3.26 所示。

将 Anaconda 的启动路径添加到系统路径 PATH 中，以方便 Anaconda 命令行的应用和 Spyder 的启动。如图 3.27 所示，输入 yes 并按回车。

Anaconda 安装完成后，需要首先关闭命令行窗口(Terminal)，再重新打开后，才能使用命令行启动 Spyder。

安装深度学习 Python 库 Keras。重新启动命令行窗口后，直接输入“conda intall keras”并按回车键后，即可进行 Keras 安装。输入 y，继续安装 Keras，如图 3.28 所示。

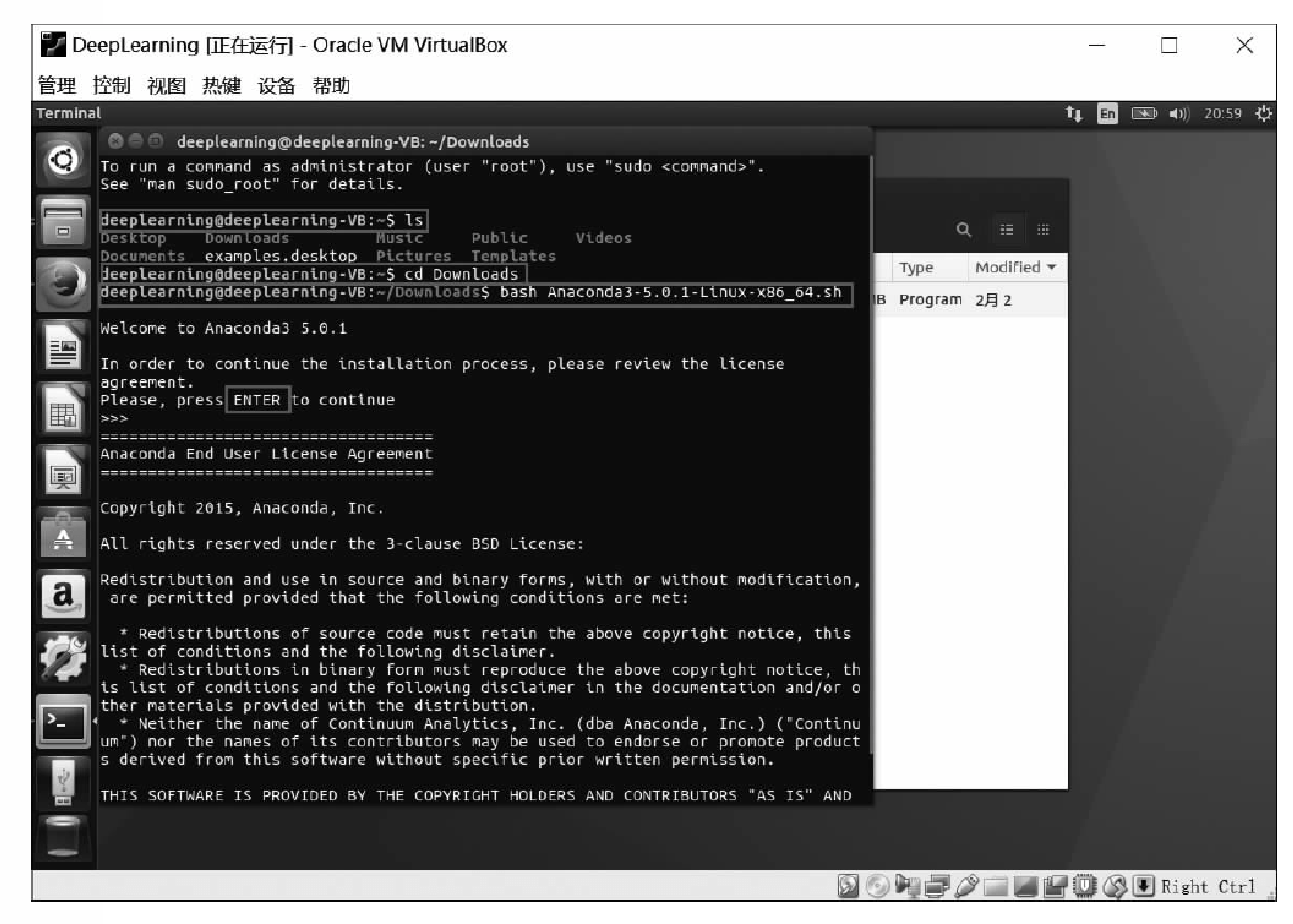


图 3.25　安装 Anaconda

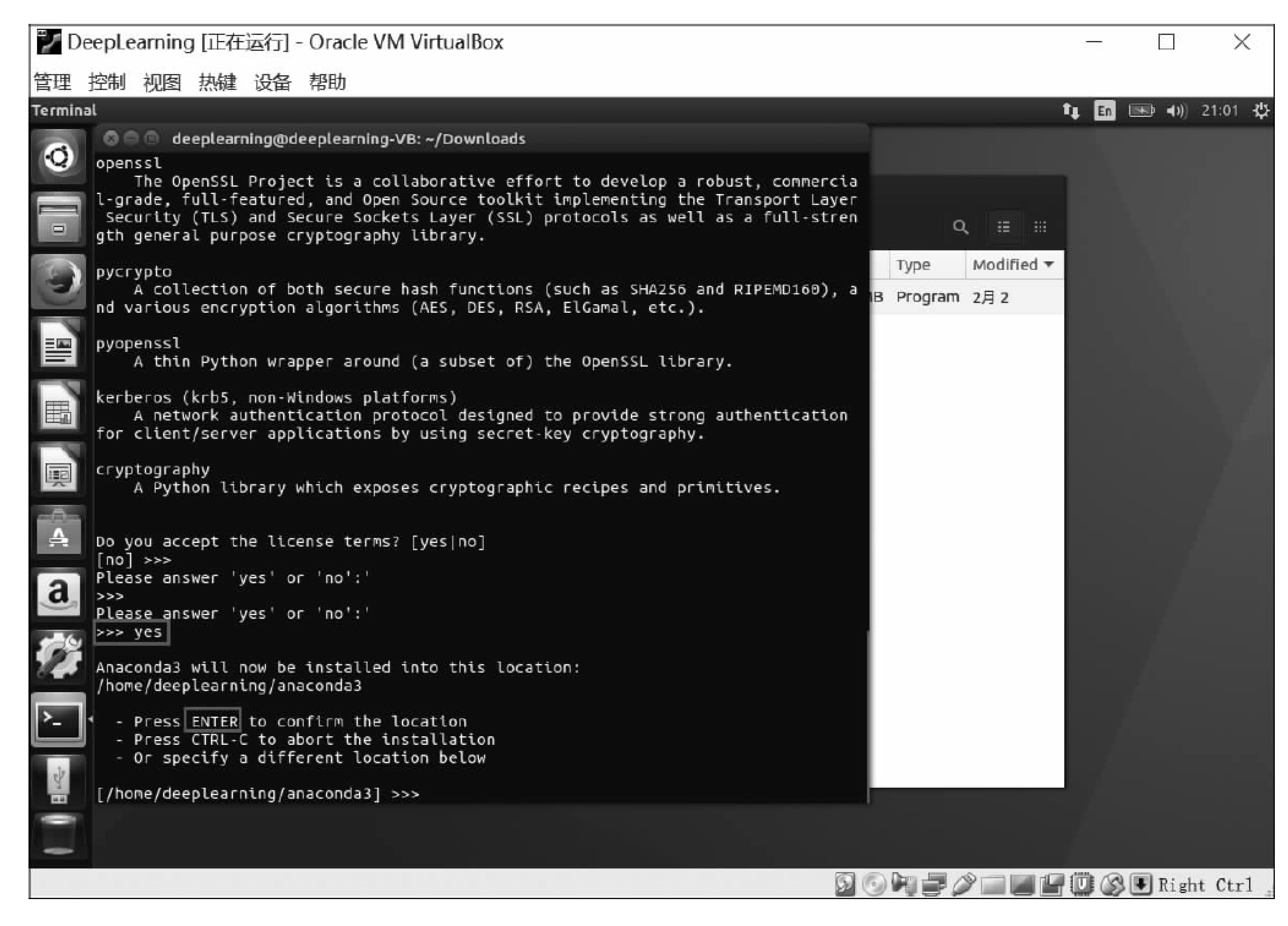


图 3.26　接受 License

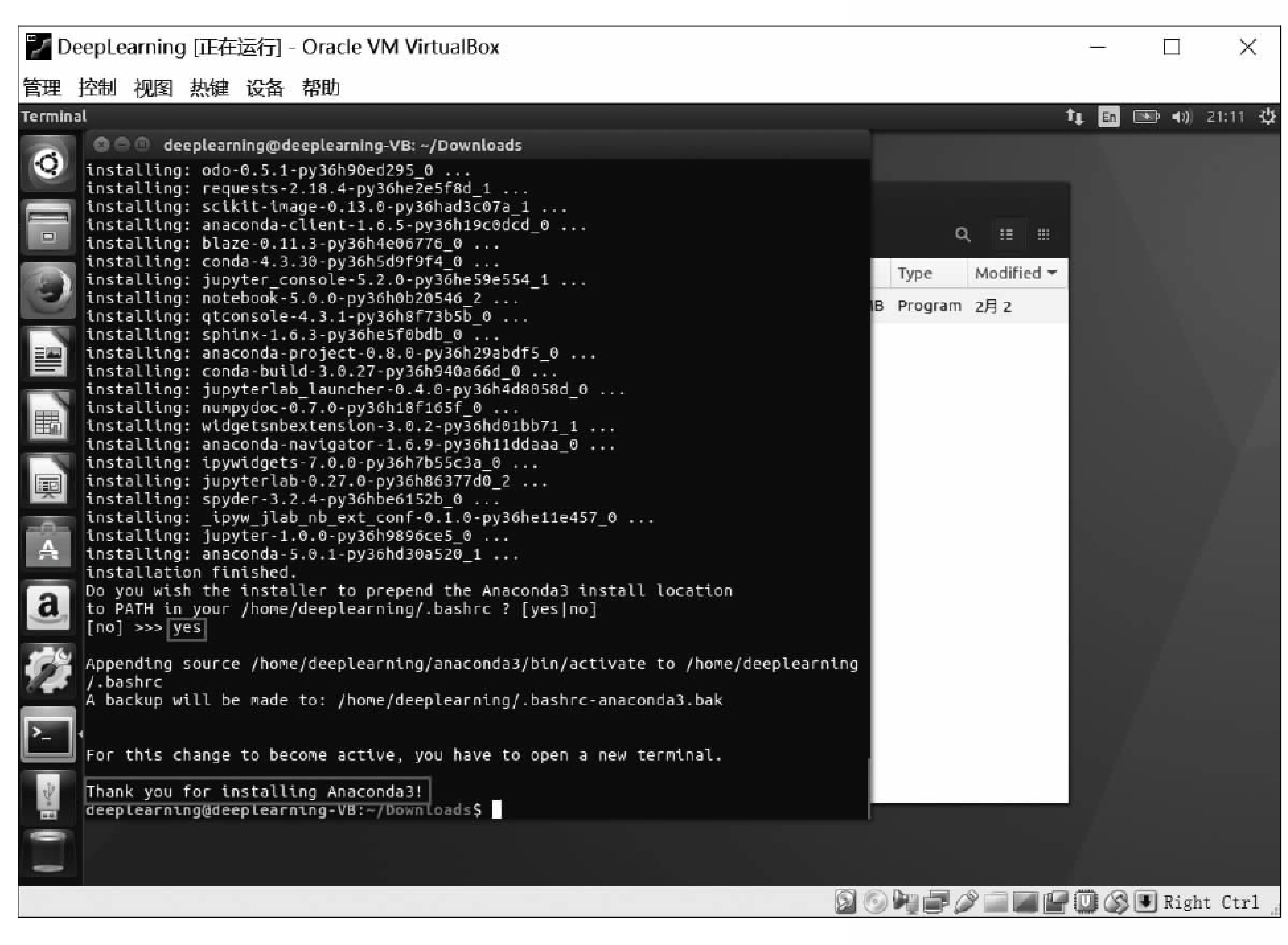


图 3.27　添加系统路径

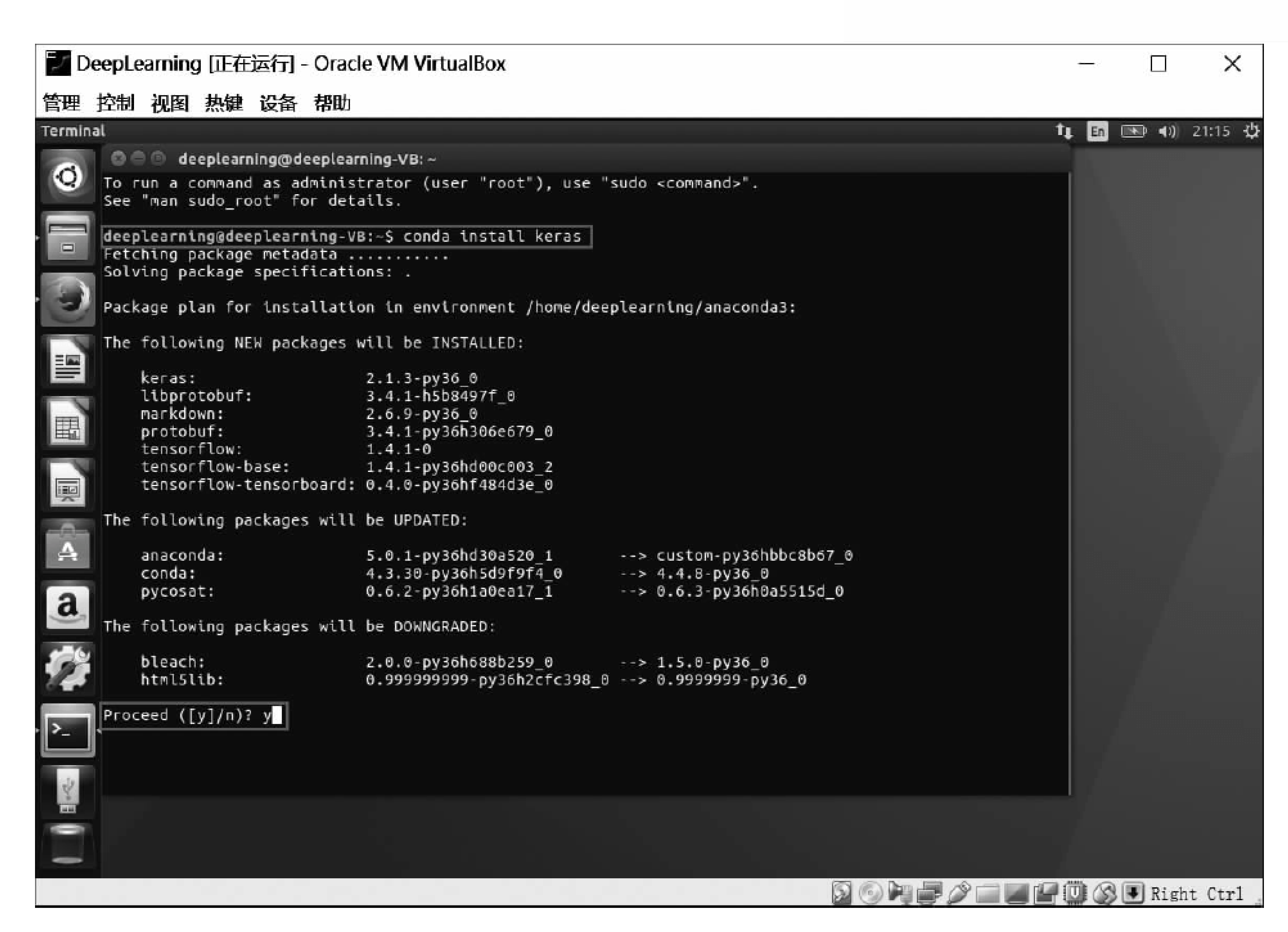


图 3.28　安装 Keras

Keras 各模块均显示 100%时，表明其安装完成。只需在命令行窗口输入 spyder 并按回车键后，即可启动 Python 集成开发环境(IDE)Spyder，如图 3.29 所示。

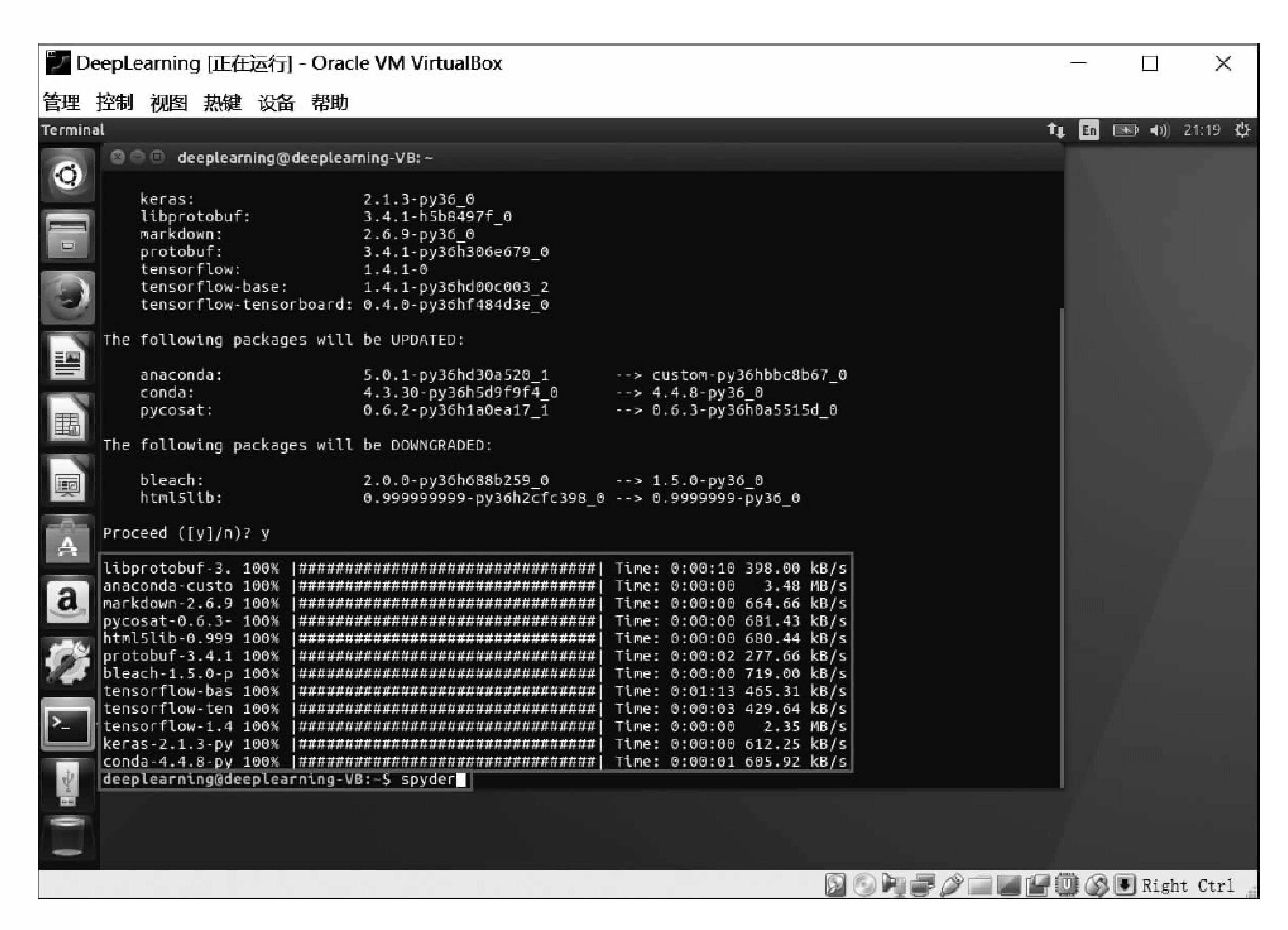


图 3.29　启动 Python 集成开发环境 Spyder

测试深度学习库 Keras 是否安装成功。在打开的 Spyder IDE 中,加载 Keras 库,如果成功则表明 Keras 安装成功。输入命令: from keras.models import Sequential,按回车键,显示 Using TensorFlow backend.表明 Keras 安装成功,如图 3.30 所示。

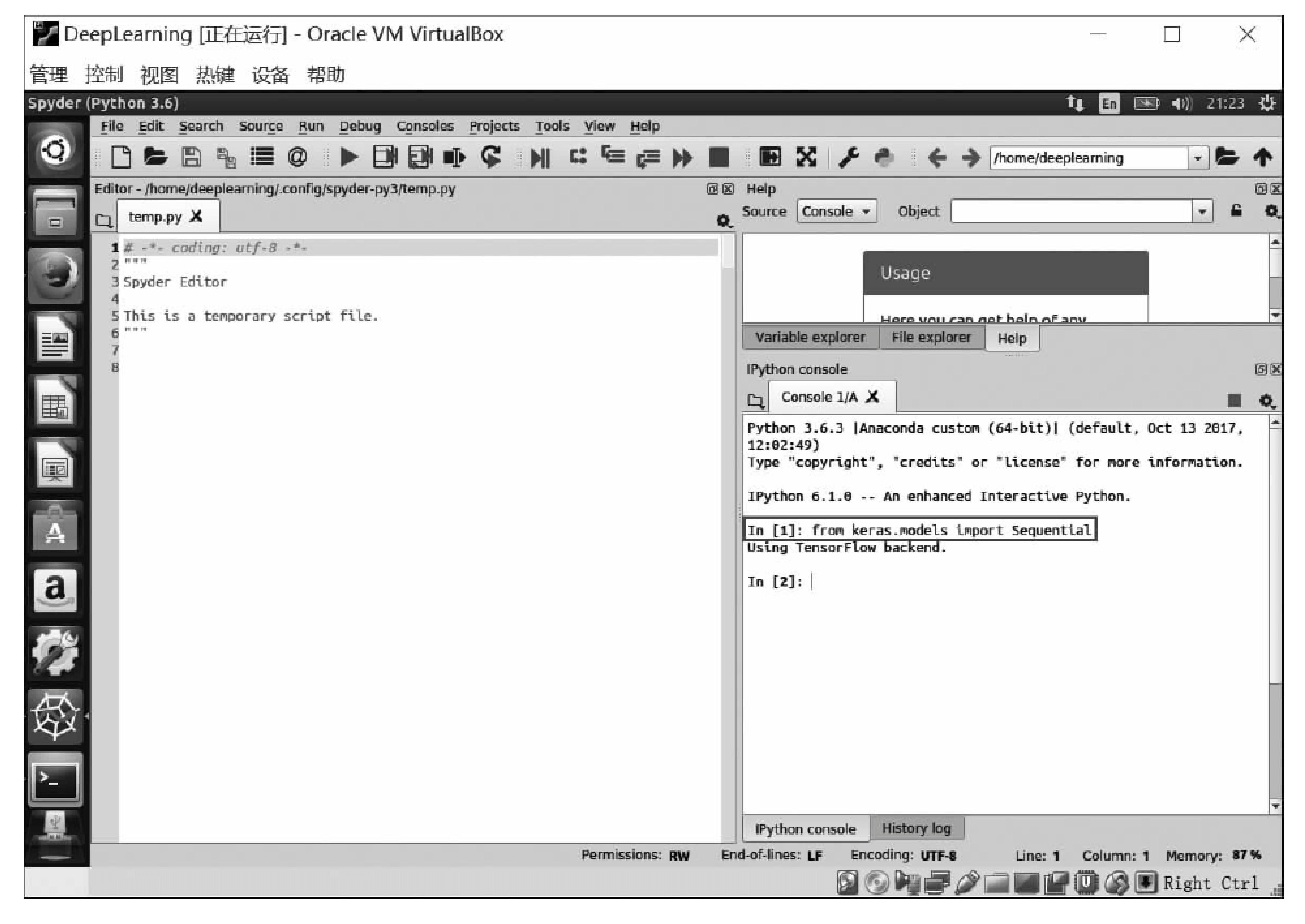


图 3.30　测试 Keras 是否安装成功

第4章 中国债券市场概况

目前，中国债券市场包括银行间市场、交易所市场和商业银行柜台交易市场。截至2017年12月31日，从存量债券余额来看，银行间市场存量债券余额占比约50%、沪深交易所市场占比约43%、柜台市场占比约7%。从存量债券余额情况来看，银行间市场和交易所市场基本处于“势均力敌”状态。

4.1 债券交易场所

由于历史的原因，中国的债券交易市场主要分为银行间市场和交易所市场两大交易场所。虽然目前这两大交易场所按存量债券余额统计计算基本各占“半壁江山”，但曾经的银行间市场占据了中国债券市场存量和交易量90%以上的份额。中国债券市场发展的历史基本可以分为五个阶段。

第一阶段从1981年到1987年，是债券市场的萌芽阶段。1981年，国债恢复发行，标志着中国债券市场的正式萌芽，1984年开始出现企业债券，1985年国有商业银行也开始发行金融债券。但是，在这一阶段并没有合法成型的债券交易机制和交易场所，债券不能进行转让和交易，债券持有人只能选择持有到期。因此，可以说这一阶段的债券市场是非常原始的，是债券市场的萌芽阶段。

第二阶段从1988年到1996年，是债券市场的起步和探索阶段。1988年，财政部和中国人民银行开始筹划建立公开的国债流通市场，1990年12月，交易所债券市场正式成立，标志着中国债券市场的发展进入一个新的历史阶段。此后，债券发行开始试行市场化，从行政摊派向承购包销及招标方式转换，1995年国债招标发行试点成功，是这一阶段债券市场发展的另一个重大突破。

第三阶段从1997年到2002年。在这一阶段，银行间债券市场异军突起，中国债券市场的框架基本形成。1997年6月，中国人民银行要求各商业银行全部退出交易所市场，同时建立了全国银行间债券市场，由外汇交易中心系统作为报价系统，由中央国债登记结算公司作为后台托管结算平台，并实行询价交易。此后，银行间债券市场逐渐发展成为所有类型机构均可参与的债券市场，并最终形成以银行间债券市场为主，交易所债券市场和商业银行柜台市场为辅的中国特色债券市场体系。

第四阶段从2003年到2015年。在这一阶段，利率市场化加速进行，债券品种丰富发展，制度建设与市场监管逐渐发展并完善。最突出的是信用产品实现大发展，如短期融资券、中期票据、无担保企业债、分离交易可转债、公司债等品种，市场规模从小到大，迅速发展。此外，衍生产品也从无到有，实现了突破，如债券远期、利率互换等产品，均在这一阶段开始出现。

第五阶段从2015年至今。这一阶段主要表现为公司债发行规模迅速增长，使得原来以银行间市场为绝对主导的中国债券市场，发展为现在银行间市场和交易所市场各占“半壁江山”的情况。这主要是因为2015年1月15日证监会颁布了公司债发行新规，使得公司债的发行方式由原来的审批制，变为公开发行核准制、私募发行备案制，且将公司债的发行主体由原来的上市公司变为非上市公司也可发行，准入门槛降得较低。因此，该阶段使得交易所市场债券发行规模迅速增长，直到2017年底存量债券的总规模跟银行间市场相比，基本相同。

经过30多年的发展，时至今日，中国债券市场已经发展成为品种多样、市场多元、存量债券余额约122万亿元的多层次债券直接融资市场，各交易场所功能分类如表4.1所示。

表4.1　中国债券市场各交易场所功能分类

市场类型	银行间债券市场	交易所债券市场	银行柜台债券市场
市场性质	场外交易	场内交易	场外交易
发行和交易券种	国债、金融债、短期融资券、定向工具、中期票据、企业债、政府支持机构债、资产支持证券、同业存单、项目收益票据、地方政府债	国债、地方政府债、金融债、政府支持机构债、企业债、公司债、资产支持证券、可交换债可转债	国债、金融债、企业债、政府支持机构债
衍生交易工具	远期利率协议、利率互换等	可分离交易可转债、普通可转债	
投资者类型	各类机构投资者	所有投资者(一部分商业银行除外)	个人和企业投资者
交易类型	现券交易、质押式回购、买断式回购、远期交易	现券交易、质押式回购	现券交易
交易方式	一对一询价交易	一对一询价交易和自动撮合	银行柜台报价
结算体制	逐笔全额结算	日终净额结算	逐笔全额结算
结算时间	T+0或T+1	T+0	T+0
债券托管机构	中债登	中证登	商业银行

4.2 信用债和利率债

债券市场主要的交易品种是债券,债券面临最主要的风险是信用风险,根据各交易品种信用风险的不同,通常将债券分为利率债和信用债。

利率债是指发行主体具有国家信用,一般不会违约的本币国家主权债券。如国债、政府支持债券、地方政府债券(因中国国情,此处暂时将政府支持债券和地方政府债券划分为利率债)、政策性金融债(国开、口行、农发)、央行票据等。发行利率债的主要目的包括为国家和地方政府财政服务、调节市场流动性和利率区间等。

信用债是指发行主体不具备本币国家主权信用,可能会违约、由企业发行并确定本息偿付现金流的债券。如企业债、公司债、短融、中票、ABS、次级债、金融债等。发行信用债的主要目的是为市场主体提供直接融资渠道,将信用风险从银行体系分散到全市场。

截至 2017 年 12 月 31 日,中国债券市场存量债券总规模约 122 万亿,按各债券类型分类统计汇总,如表 4.2 所示。可见,利率债占据约 3/4 的存量债券余额,且其中占比最高的是国债。信用债仅占存量债券余额的 1/4,在信用债中占比最大的债券品种为企业债、公司债和中期票据。

表 4.2 债券类型分类统计汇总

类型	债券分类	余额/亿元	占比/%
利率债	国债	403 186.87	33.14
	地方政府债	371 721.35	30.55
	政策银行债	126 009.56	10.36
信用债	定向工具	19 499.61	1.60
	国际机构债	230.00	0.02
	保险公司债	1 377.50	0.11
	集合票据	1.86	0.00
	集合企业债	204.20	0.02
	交易商协会 ABN	537.08	0.04
	可交换债	1 823.79	0.15
	可转债	1 198.18	0.10
	其他金融机构债	3 453.80	0.28
	商业银行次级债券	20 277.62	1.67

续表

类型	债券分类	余额/亿元	占比/%
信用债	商业银行债	10 154.20	0.83
	私募债	23 608.58	1.94
	政府支持机构债	14 700.00	1.21
	项目收益票据	95.60	0.01
	一般短期融资券	3 435.40	0.28
	一般公司债	26 576.01	2.18
	一般企业债	53 826.87	4.42
	一般中期票据	47 345.71	3.89
	银监会主管 ABS	6 291.04	0.52
	证监会主管 ABS	8 774.09	0.72
	证券公司短期融资券	50.00	0.00
	证券公司债	12 785.75	1.05
	超短期融资债券	8 651.30	0.71
其他	同业存单	50 756.30	4.17
合　　计		1 216 572.27	100.00

第5章 信用债投资面临的困难和解决方案

中国市场进行信用债投资时，面临着许多客观实际的困难，如外部评级区分能力太差、对投资基本没有参考价值，财务粉饰现象比较常见、财务报表无法真实有效地反映企业真实的还款能力，财报披露不及时、财报披露为发债服务，发债成功后不再披露财务信息等，这些客观存在的实际困难都给信用债投资者带来很大的困难。本章重点分析这些困难，并给出相应的解决方案。

5.1 信用债分析面临的主要困难

由于我国市场经济体制建设起步较晚，全社会信用体系建设也不健全，上市公司财务欺诈现象时有发生。因此，我们在做信用债投资时不可避免地面临一些无法克服的实际困难，就笔者近10年的研究分析来看，这些困难主要表现为以下几点。

第一，外部评级区分能力太差，对信用债投资几乎没有参考价值。截至2018年2月底，资本市场存量有评级信用债约17 000只，这些债券的主体评级分布情况，如图5.1所示。可见，外部评级非常集中在前四个等级，其中AAA占比约29%、AA+占比约22%、AA占比约37%、AA－占比9%。由于+/－符号仅代表略高或略低于本等级，表示微调，并未表示较大等级间的差异，因此合并统计的情况下AAA占比约29%，AA(含AA+、AA－)占比约68%，这两个等级合计占比约97%。

作为对比，我们通过穆迪官网找出所有的穆迪全球样本主体评级数据，绘制如图5.2所示的分布图。和中国市场主体评级分布图对比，我们可以清晰地发现，穆迪全球样本的主体评级分布非常分散，非常类似于右偏的Beta分布，且高信用等级(如Aaa和Aa)的占比极少，其中Aaa和Aa两个最高信用等级的合计占比只有约1.3%。

那么，被评主体实际违约情况跟评级符号所代表的含义是否一致呢？我们先来看下国内评级机构对评级符号的定义，如表5.1所示。这个评级符号的定义，跟穆迪评级符号的定义基本一致。也可基本认为国内外部评级机构对信用等级符号的定义是从穆迪等国外评级机构翻译过来的。



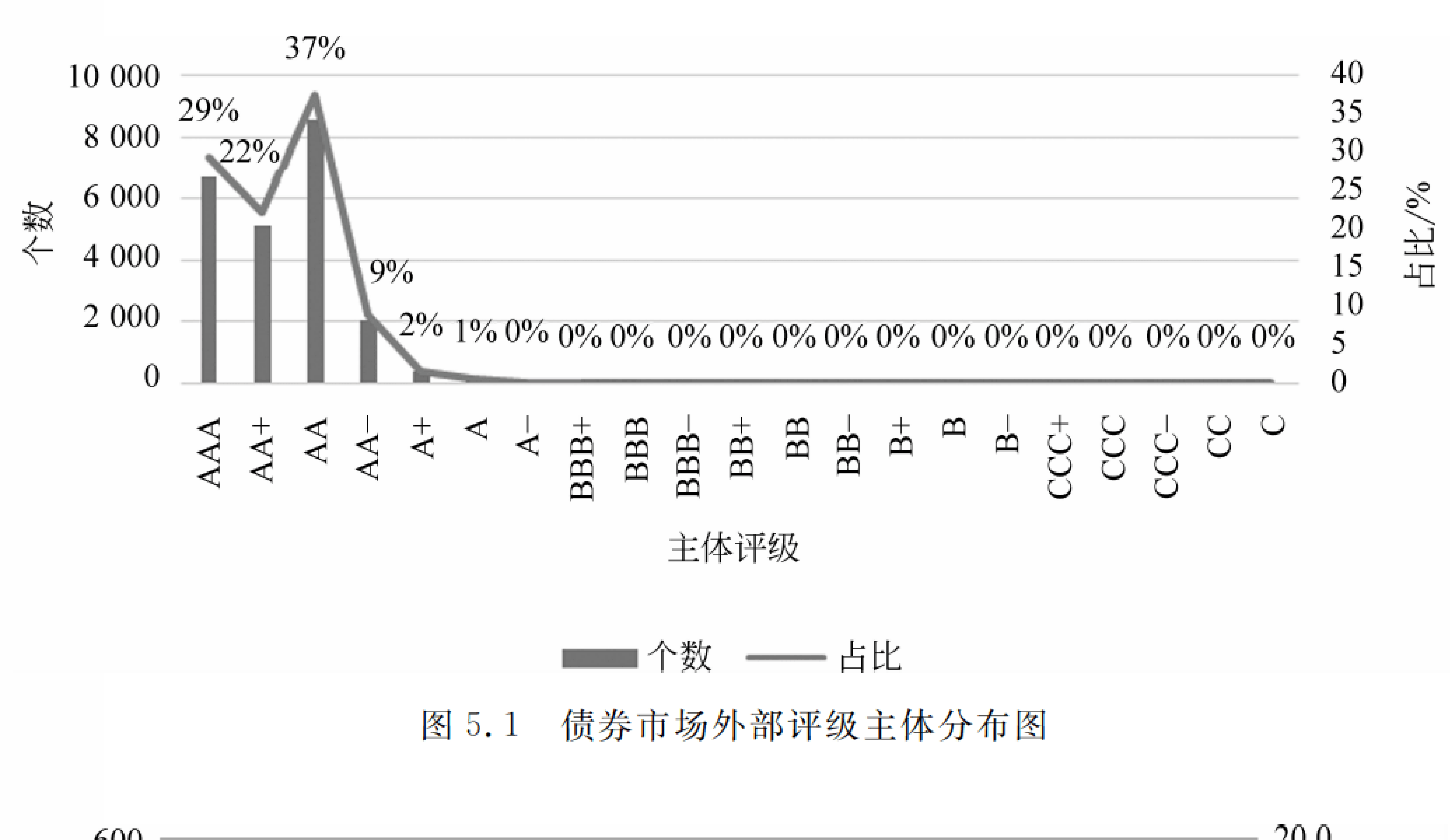


图 5.1　债券市场外部评级主体分布图

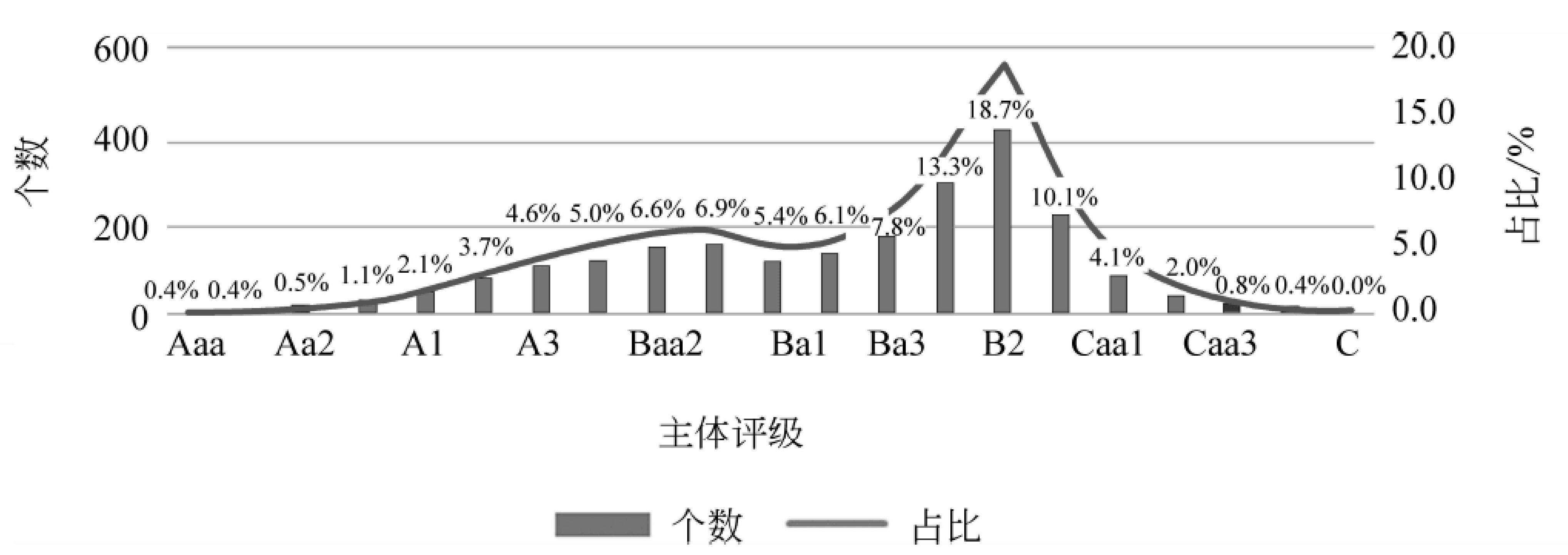


图 5.2　穆迪全球样本主体评级分布图

表 5.1　评级符号的定义

序号	评级符号	定　　义
1	AAA	受评对象偿还债务的能力极强，基本不受不利经济环境的影响，违约风险极低
2	AA	受评对象偿还债务的能力很强，受不利经济环境的影响较小，违约风险很低
3	A	受评对象偿还债务的能力较强，较易受不利经济环境的影响，违约风险较低
4	BBB	受评对象偿还债务的能力一般，受不利经济环境的影响较大，违约风险一般
5	BB	受评对象偿还债务的能力较弱，受不利经济环境的影响很大，有较高违约风险
6	B	受评对象偿还债务的能力较大地依赖于良好的经济环境，违约风险很高
7	CCC	受评对象偿还债务的能力极度依赖于良好的经济环境，违约风险极高

续表

序号	评级符号	定　　义
8	CC	受评对象在破产或重组时可获得保护较小，基本不能保证偿还债务
9	C	受评对象不能偿还债务

注：除AAA和CCC级以下等级外，每一个信用等级可用“＋”“－”符号进行微调，表示略高或略低于本等级。

我们再来看下资本市场实际违约情况跟评级机构的评级预测是否一致。先看下国内市场，截至2018年2月底，国内资本市场有164只债券发生了实质性违约，其中有119只债券是有评级的，这些有评级债券违约前1年主体评级分布情况，如图5.3所示。可见，违约债券的主体评级主要集中在AA等级（含AA＋和AA－），约占所有违约债券的90％。

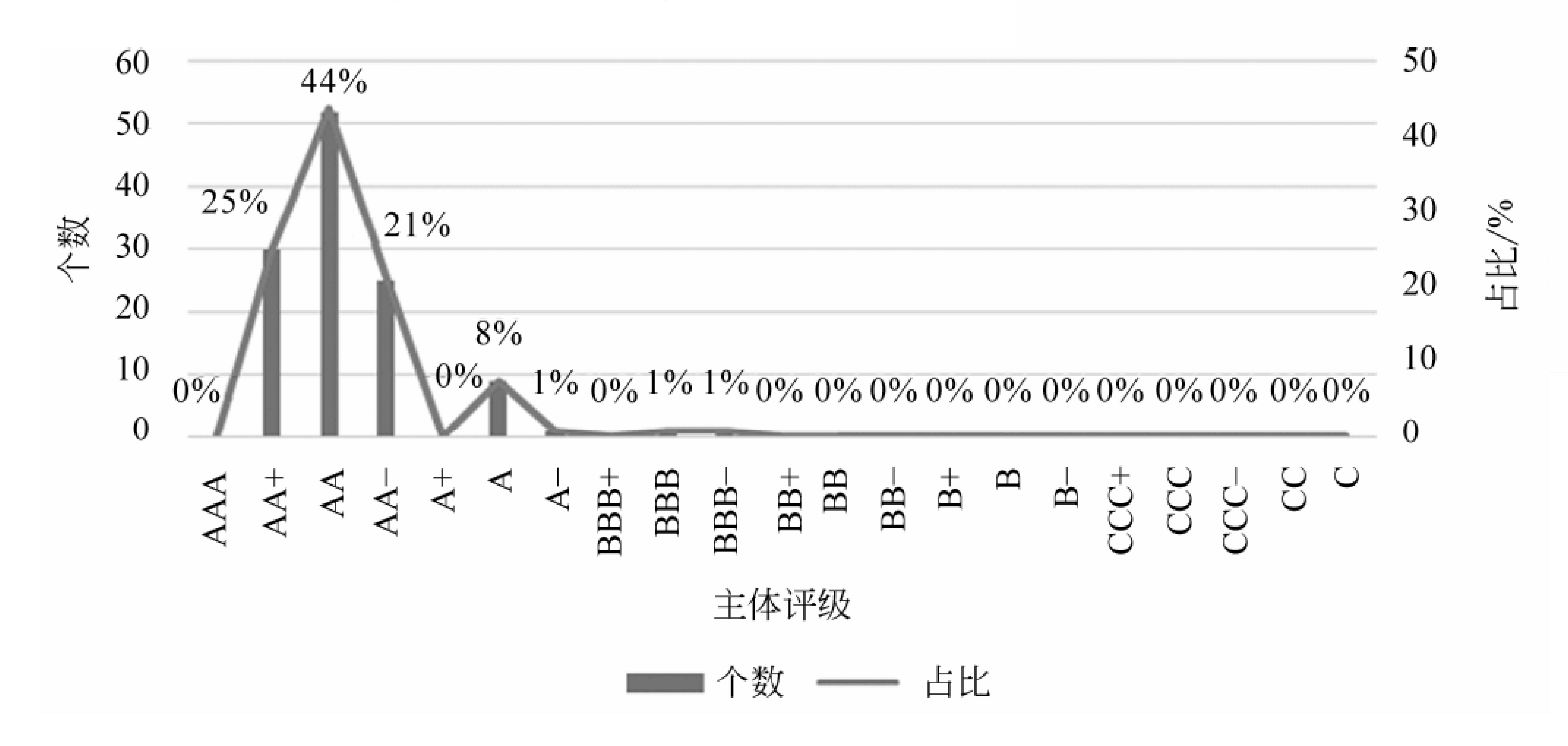


图5.3　已违约债券主体评级分布图

再看国际市场的情况，查阅穆迪官网我们可以找到所有被评机构历史发生违约情况变化曲线图，如图5.4所示。虽然该曲线只有3条，却代表多重深层次含义。

其中，最下面那条曲线，代表曾经被穆迪评为投资级（BBB－及以上属于投资级）的主体发生违约的情况。可见，在正常市场情况下，被穆迪评为投资级的主体，基本没有发生过违约。只有在爆发全球性经济或金融危机时，投资级主体才会发生违约，如2002年前后的美国高科技泡沫破灭和2008年发生的次贷危机均造成了投资级主体发生违约的情况。

最上面那条曲线，代表曾经被穆迪评为投机级（BB＋及以下属于投机级）的主体发生违约的情况。可见，在正常市场情况下，被穆迪评为投机级的主体会偶尔发生违约。但在极端情况发生时，投机级主体发生违约的情况会快速增加。

中间那条曲线代表的是所有被穆迪评级的主体发生违约的情况，可见其发

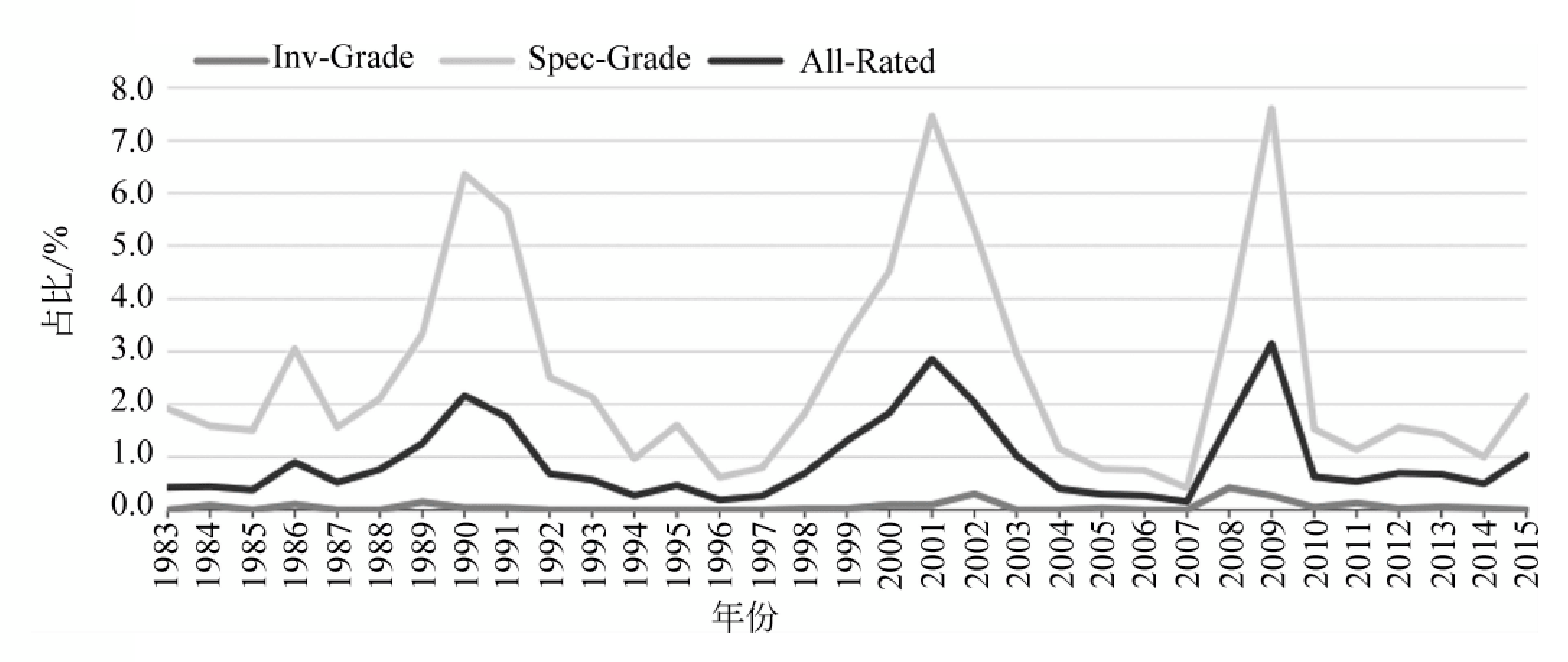


图 5.4 穆迪全球样本主体评级与违约情况变化曲线图

生违约的情况跟投机级主体发生违约的情况强相关。

通过以上分析，我们认为，被穆迪评为投资级的主体在正常市场情况下，基本不会发生违约，只有在极端市场情况下，如经济危机发生时，才可能发生违约。而被穆迪评为投机级的主体在正常市场情况下偶尔会发生违约，在极端市场情况下发生违约的主体快速增加。

穆迪评级发生实际违约的情况，跟其评级符号所代表的含义基本一致，而国内外部评级的评级符号跟其发生实际违约的情况非常不一致。例如，国内外部评级机构将 AA 主体的信用描述为“受评对象偿还债务的能力很强，受不利经济环境的影响较小，违约风险很低”。可是，实际违约事件 90％发生在 AA（含 AA＋和 AA－）的受评主体中。

最后，我们看下国内评级机构和穆迪的评级结果是否可用于信用风险定价。笔者通过彭博资讯（Bloomberg）导出所有的中国公司在美国市场发行过的美元债券，并获取它们被穆迪给予的评级结果和债券发行的票面利率。通过统计汇总相同信用等级发行债券票面利率的最小值、最大值、中位数和平均数，绘制如图 5.5 所示的雷达图。可见，穆迪的评级结果跟反映信用风险定价的票面利率直接强相关，即被穆迪评为高信用等级（如投资级）的主体发行债券的票面利率比较低，而被穆迪评为低信用等级（如投机级）的主体发行债券的票面利率相对投资级来说比较高。表现出明显的分层特性，这也说明穆迪的评级结果可用于信用风险定价。

用同样的方法，我们也统计了国内存量债券相同信用等级的发债主体，所发行债券票面利率的最小值、最大值、中位数和平均数，并绘制图 5.6 所示的雷达图。可见，图中的四个四边形无明显的分层特性，与图 5.5 所示的雷达图有明显的区别。这充分说明了国内的外部评级是不能用作风险定价的，或者国内债券发行主体的融资成本跟评级结果关系不大。

第二，财务数据对违约的影响不显著。传统的信用债投资分析是建立在主

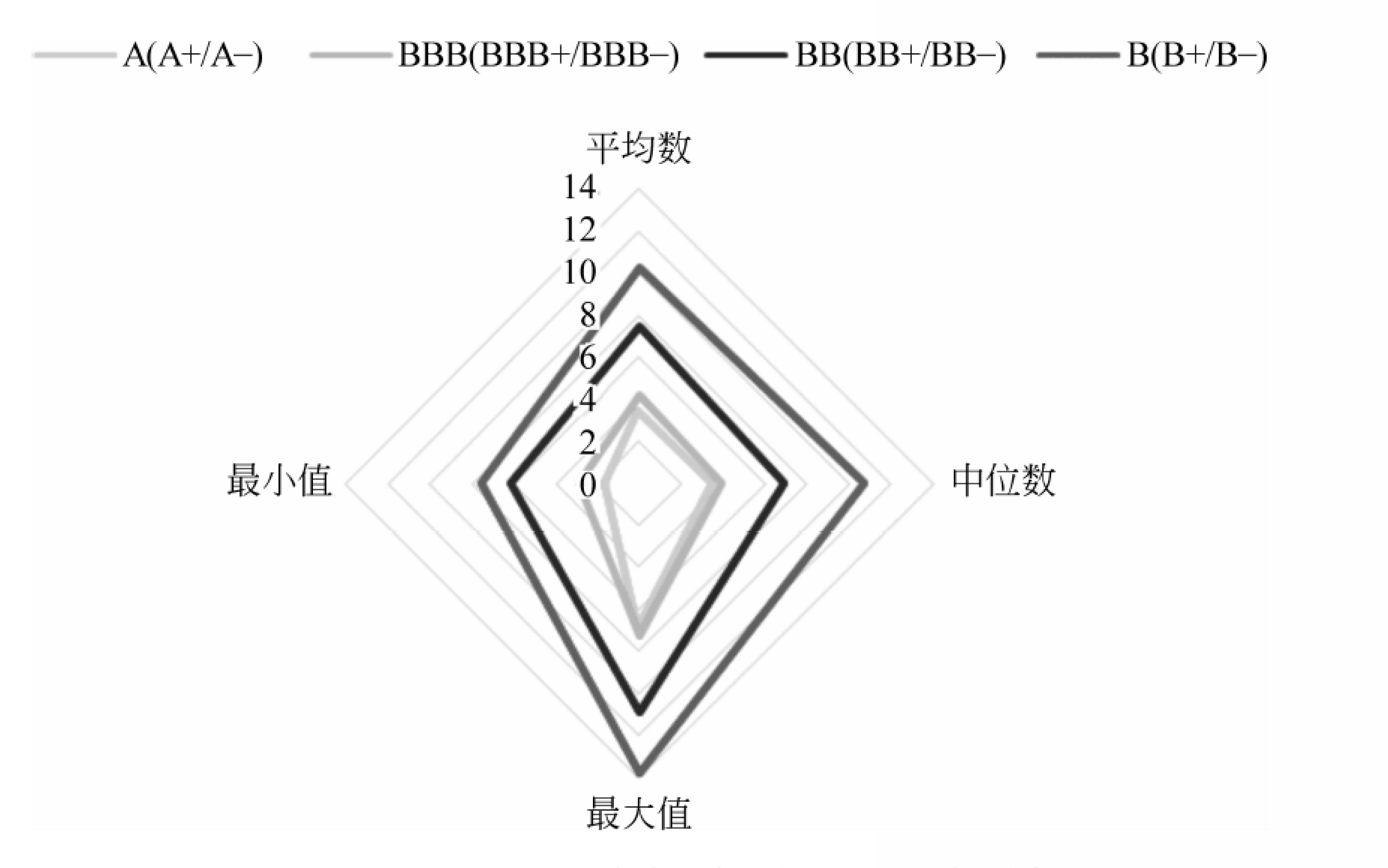


图 5.5　中国公司发行美元债券,穆迪评级和融资利率关系图

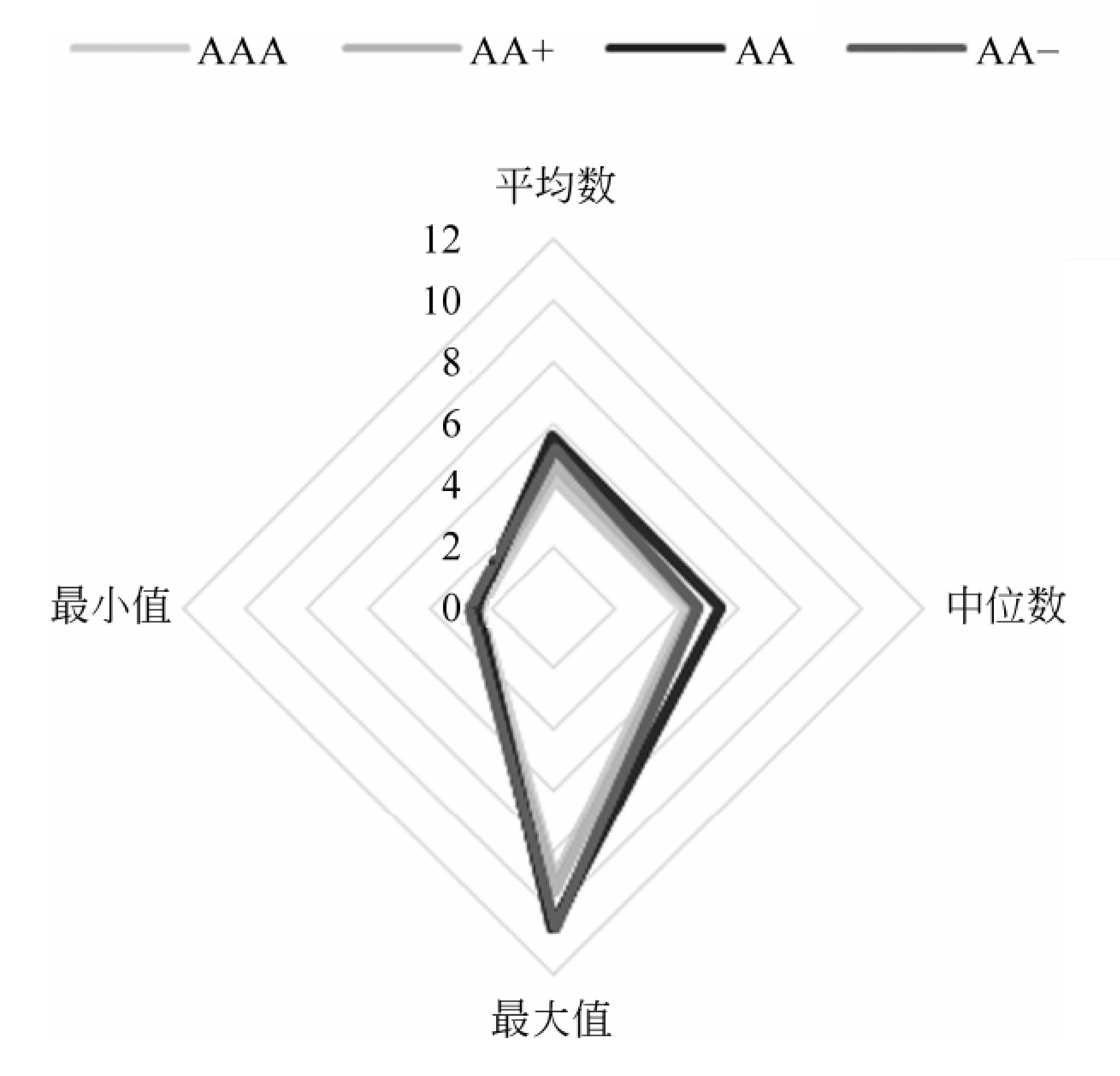
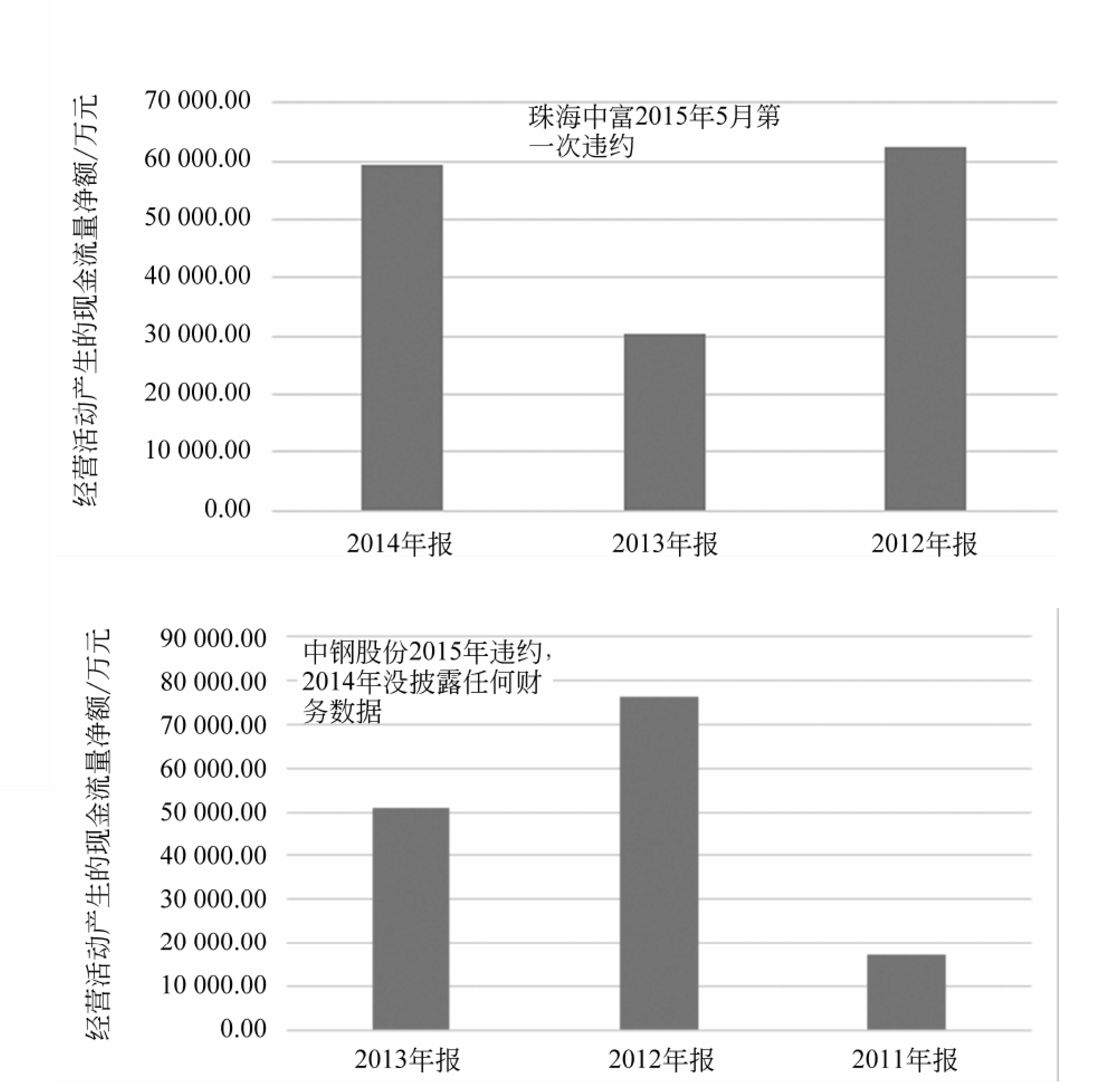


图 5.6　国内外部评级和融资利率关系图

要分析企业财务数据的基础上,可是现阶段企业粉饰财报的现象时有发生,笔者统计了已经违约企业的财务数据,发现这些企业在违约前的现金流和盈利状况还基本是“良好的”,部分违约公司的经营性现金流净额在其违约前 3 年的情况,如图 5.7 所示。经营性现金流净额为正,说明这家企业的主营业务经营良好,可是实际情况却是经营性现金流净额“良好”的企业发生了违约。

用同样的方法,笔者统计了存量 3 542 只民营企业信用债对应的 914 家发行主体,并从中选出连续 3 年亏损的 5 家企业和连续两年亏损的 20 家企业,发现这 25 家企业均未发生违约事件,统计结果如图 5.8 所示。



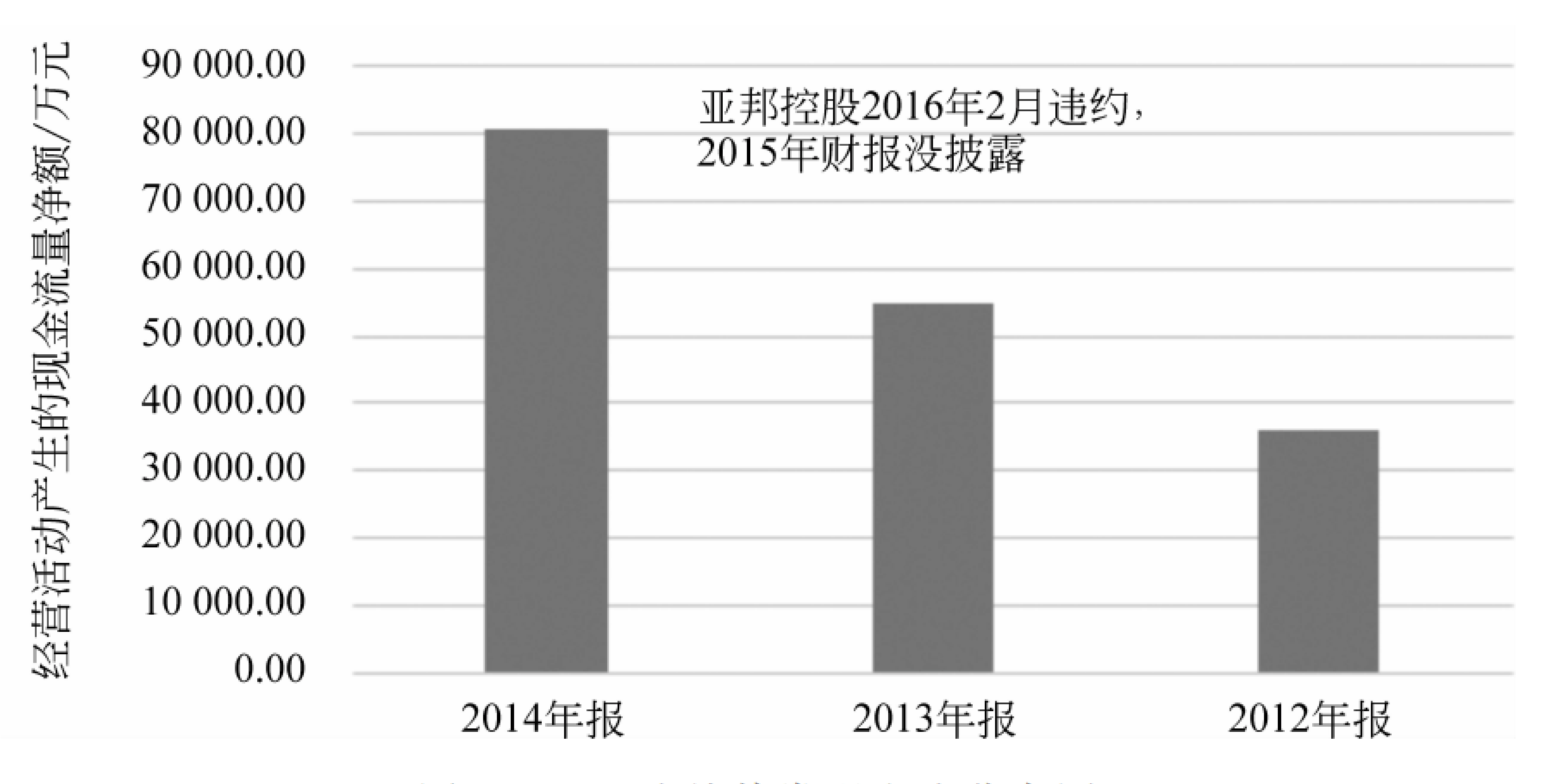
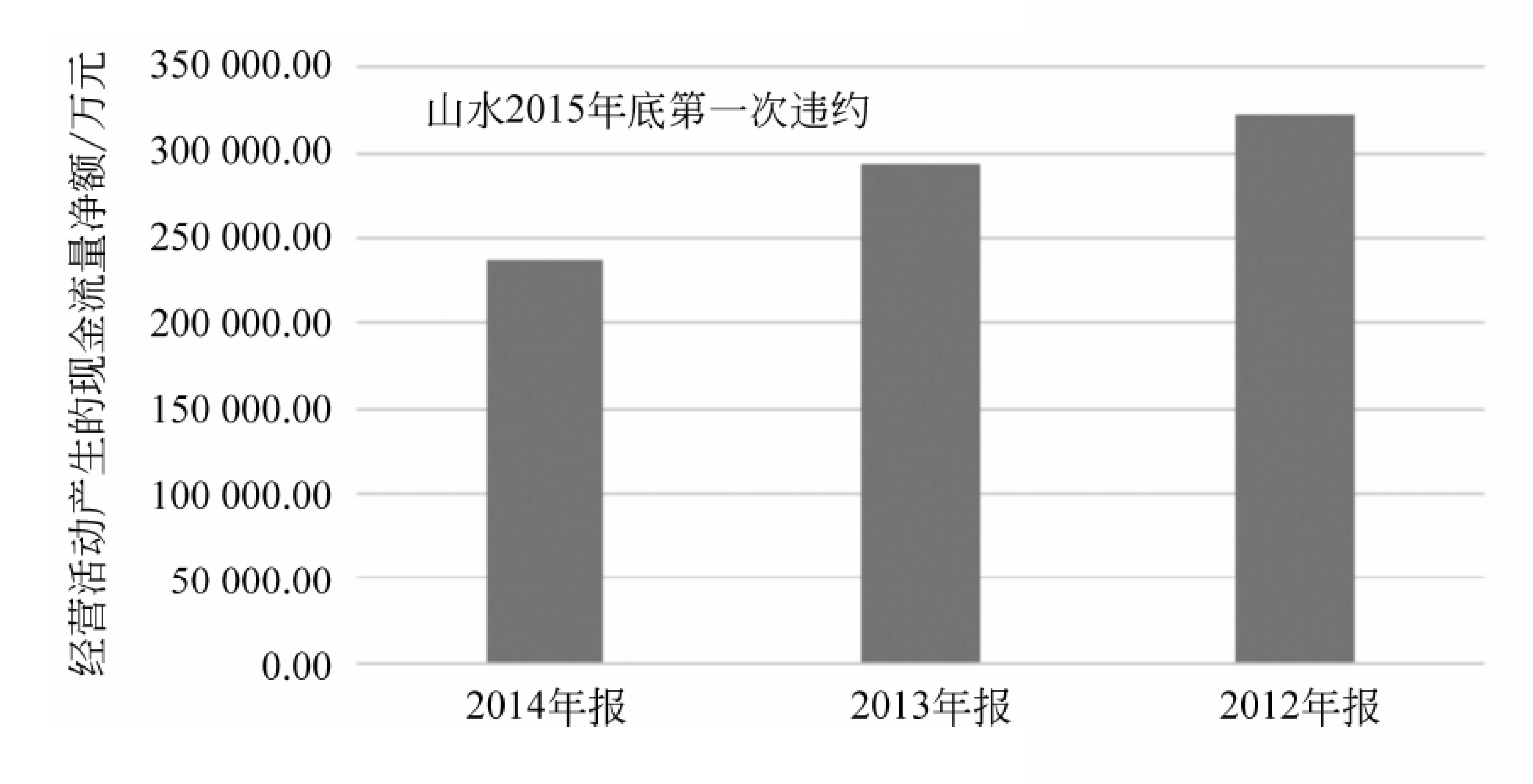
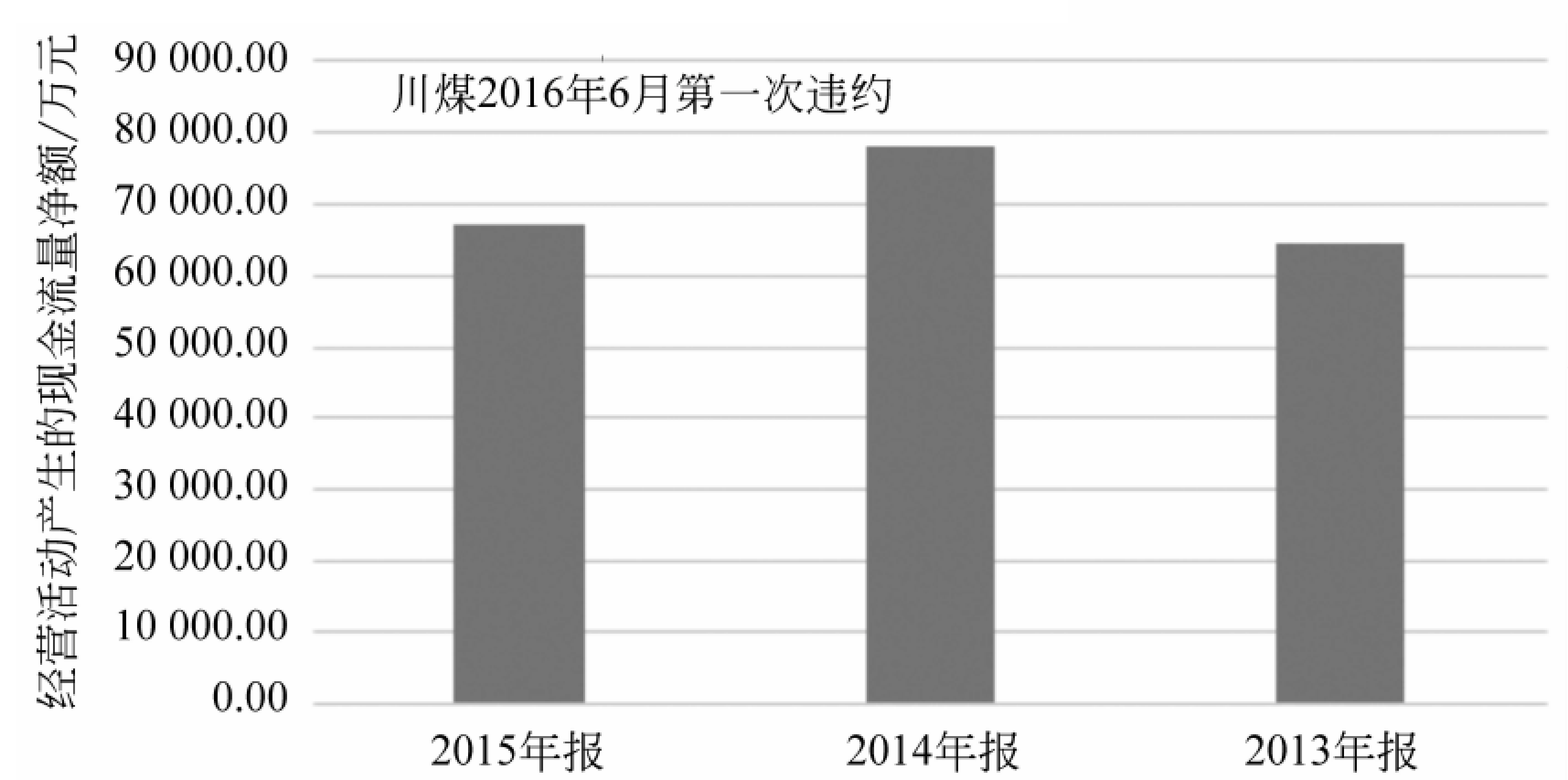


图 5.7　已违约债券现金流分布图


山水2015年底第一次违约
经营活动产生的现金流量净额/万元
350 000.00
300 000.00
250 000.00
200 000.00
150 000.00
100 000.00
50 000.00
0.00
2014年报
2013年报
2012年报


川煤2016年6月第一次违约
经营活动产生的现金流量净额/万元
90 000.00
80 000.00
70 000.00
60 000.00
50 000.00
40 000.00
30 000.00
20 000.00
10 000.00
0.00
2015年报
2014年报
2013年报

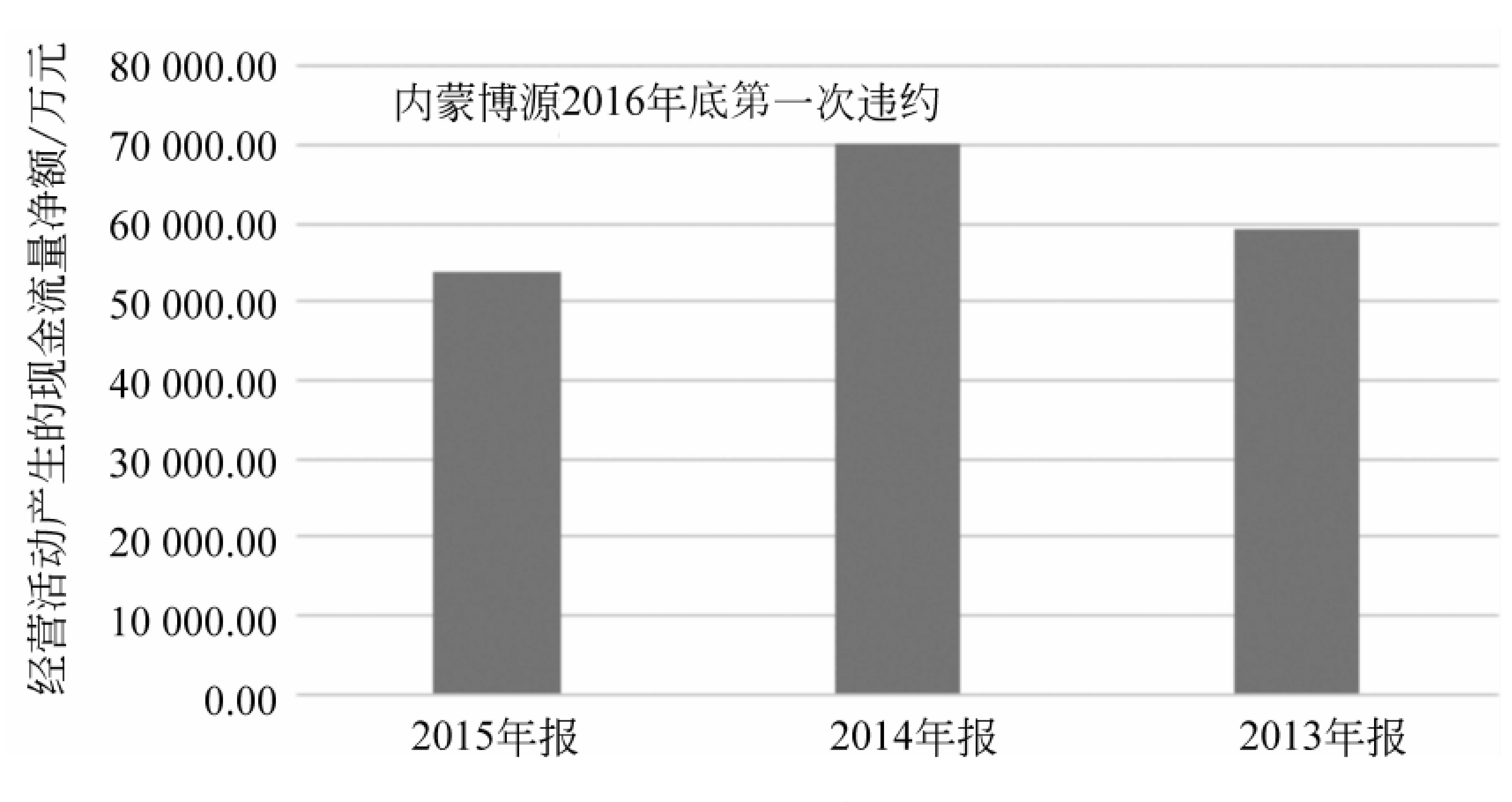

内蒙博源2016年底第一次违约
经营活动产生的现金流量净额/万元
80 000.00
70 000.00
60 000.00
50 000.00
40 000.00
30 000.00
20 000.00
10 000.00
0.00
2015年报
2014年报
2013年报

图 5.7 （续）

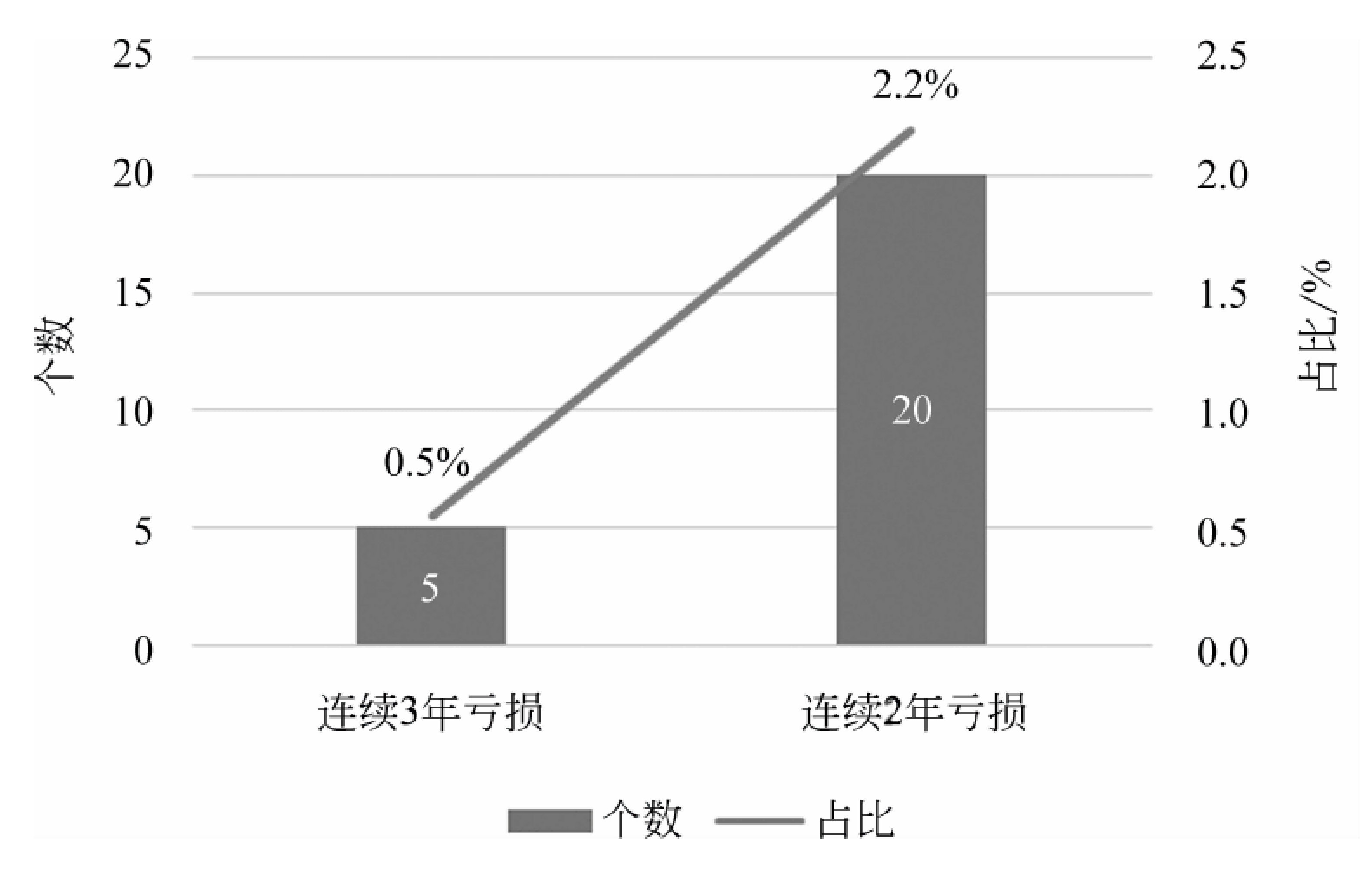


图 5.8　亏损企业未违约统计图

第三，发债主体的财务信息披露不及时。非上市公司的财务披露多数为发债融资服务，融资完成后财务信息披露意愿通常较差。笔者通过 Wind 提取所有信用债，并剔除保险公司债、同业存单、政策性银行债、政府支持机构债券后，共计剩余 17590 只债券，这些债券中无最新季报披露的企业约占 34%，如图 5.9 所示。

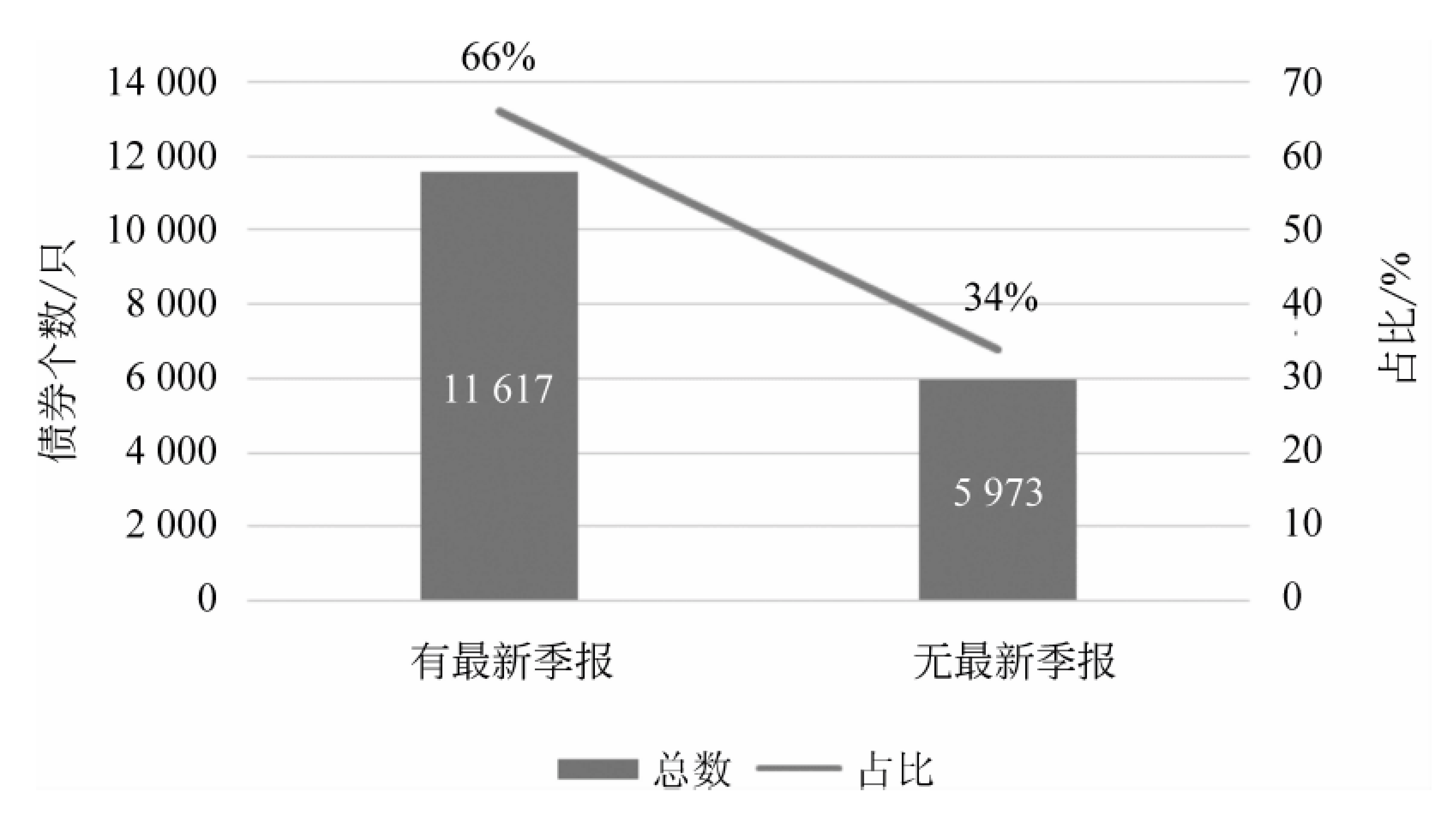


图 5.9　全市场信用债财报披露及时性统计图

用同样的方法，统计已经违约的 164 只债券。如果我们将“无最新季报披露”定义为违约前 120 天以上未披露财务信息，将“有最新季报披露”定义为违约前 120 天以内披露了财务信息，则统计发现已经违约的 164 只债券中，有 19% 的债券发行主体有最新季报披露，有 81% 的债券发行主体无最新季报披露，如图 5.10 所示。



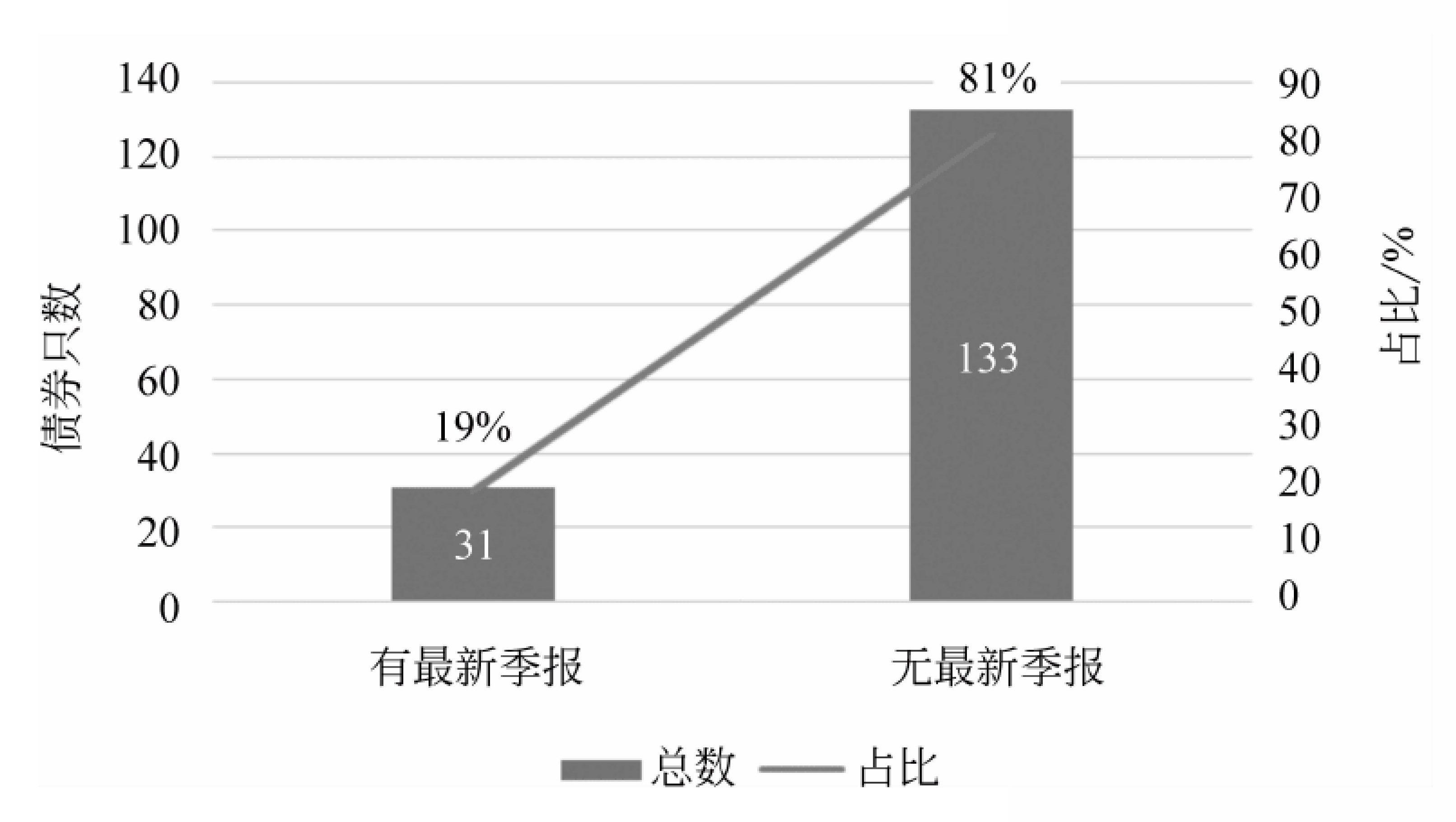


图 5.10　已违约债券财报披露及时性统计图

5.2　解决方案

根据 5.1 节的分析，在国内资本市场的现状下，评估信用债发行主体的真正价值，遇到很多客观存在且短期内无法解决的问题，我们总结如下。

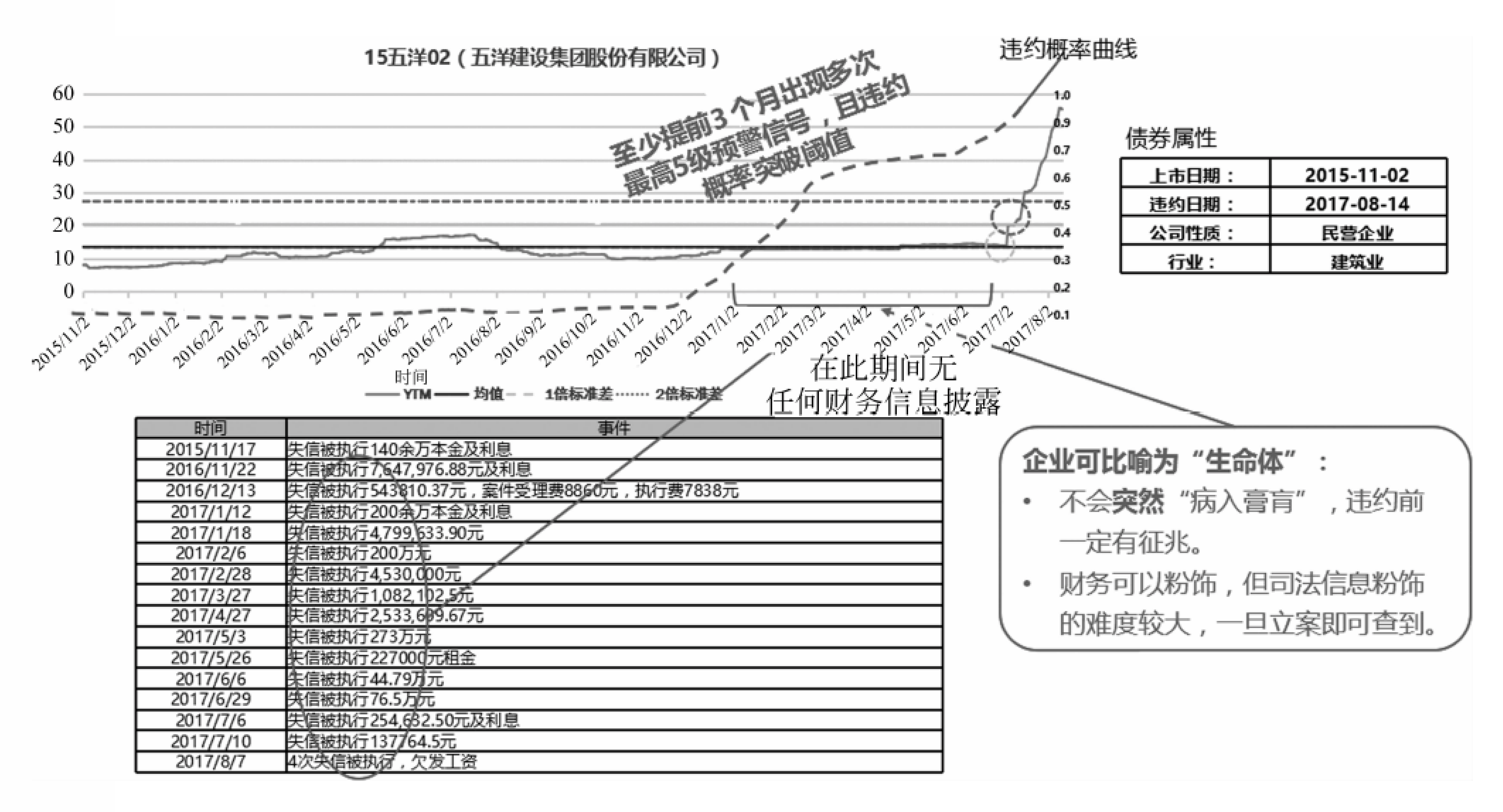
(1) 企业可比喻为一个“生命体”，任何一个生命体都不会突然“病入膏肓”，就如同任何一个今天还非常健康的人，不可能明天就突然患癌症去世，在患癌症前肯定有一些征兆。企业违约也是同样的道理，任何一家今天现金流和盈利都良好的企业，也不可能明天就突然违约了。财务报表是一家企业是否良好的“体检表”，可是将这个“体检表”用于“体检”国内企业的“健康状况”时，经常会出现误判。这主要是因为目前国内企业粉饰财报的现象较常发生，用财务报表这张企业的“体检表”来判断企业的健康状况基本处于失灵状态，信用债投资者基本有这么一个共识，那就是财报中能发现问题的企业，早就违约了；违约的企业基本都是财报中没有发现问题的。

(2) 财报披露不及时，很多企业的财报披露是为企业的发债融资服务的，融资成功后，财报披露意愿较差。

由于这些客观存在的困难，传统的基于财务分析的企业信用风险评估方法，目前基本不能用于信用债投资分析。笔者经过近 10 年来对中国市场现状的深度研究，发现场外非财务数据更能真实反映企业自身的还款能力和财务水平。我们先举两个例子。

第一个是五洋债，在其违约前 1 年左右的时间内，基本没有任何财务信息披露。此时，如果我们采用传统的基于企业财务数据的企业信用风险评估方法，则很难获得该企业的真实财务水平。但通过查询跟五洋建设集团相关的司法数据，我们在这未披露任何财务数据的 1 年左右的时间里，发现这家公司因欠款被告了很多次，且在 2017 年初以来累计有超过 15 次被法院列为“失信被执行人”，

如图 5.11 所示。



时间	事件
2015/11/17	失信被执行140余万本金及利息
2016/11/22	失信被执行7,647,976.88元及利息
2016/12/13	失信被执行543810.37元，案件受理费8860元，执行费7838元
2017/1/12	失信被执行200余万本金及利息
2017/1/18	失信被执行4,799,633.90元
2017/2/6	失信被执行200万元
2017/2/28	失信被执行4,530,000元
2017/3/27	失信被执行1,082,102.5元
2017/4/27	失信被执行2,533,609.67元
2017/5/3	失信被执行273万元
2017/5/26	失信被执行227000元租金
2017/6/6	失信被执行44.79万元
2017/6/29	失信被执行76.5万元
2017/7/6	失信被执行254,682.50元及利息
2017/7/10	失信被执行137764.5元
2017/8/7	4次失信被执行，欠发工资

图 5.11　五洋债违约前的司法信息

第二个是被认为不会发生违约的中央国有企业——中国中钢股份有限公司。同样，在中钢违约前 1 年左右的时间里，也没有披露任何财务信息。但通过查询跟中钢有关的司法信息，我们发现这家公司在违约前 1 年左右的时间里，出现 3 次被银行起诉并冻结存款或申请诉前资产保全的案例，并存在其子公司因欠款被起诉的案件，如图 5.12 所示。

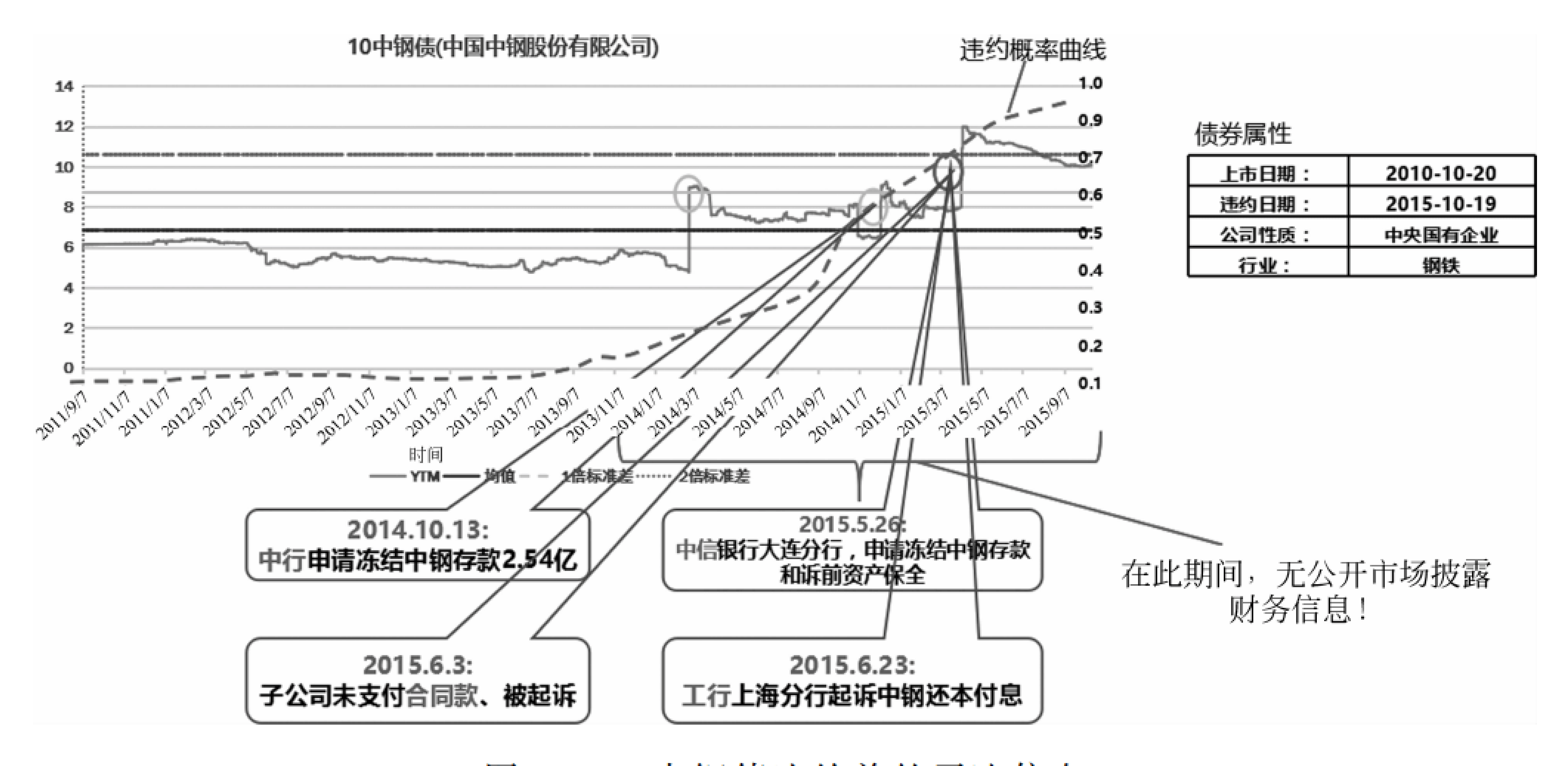


图 5.12　中钢债违约前的司法信息

企业可以比较方便地粉饰自己的财报，但被起诉案件一旦受理，即可在中国裁判文书网（http://wenshu.court.gov.cn/）查到，且无法造假。如果被诉案件中涉及较多的赔款信息，说明这家企业财务状况较差。

由于国内债券市场的流动性较差，当一家企业被法院列为失信被执行人后

距离问题爆发和债券违约的时间点，往往比较临近了。此时，即使作出了正确的预测也较难卖出持有的债券。根据国内法院关于处理涉诉且需要赔款的流程，如图 5.13 所示，获取企业被告且需要赔款时的涉诉信息，就可以比较及时地判断企业未来违约的趋势变化，并作出相应的判断。

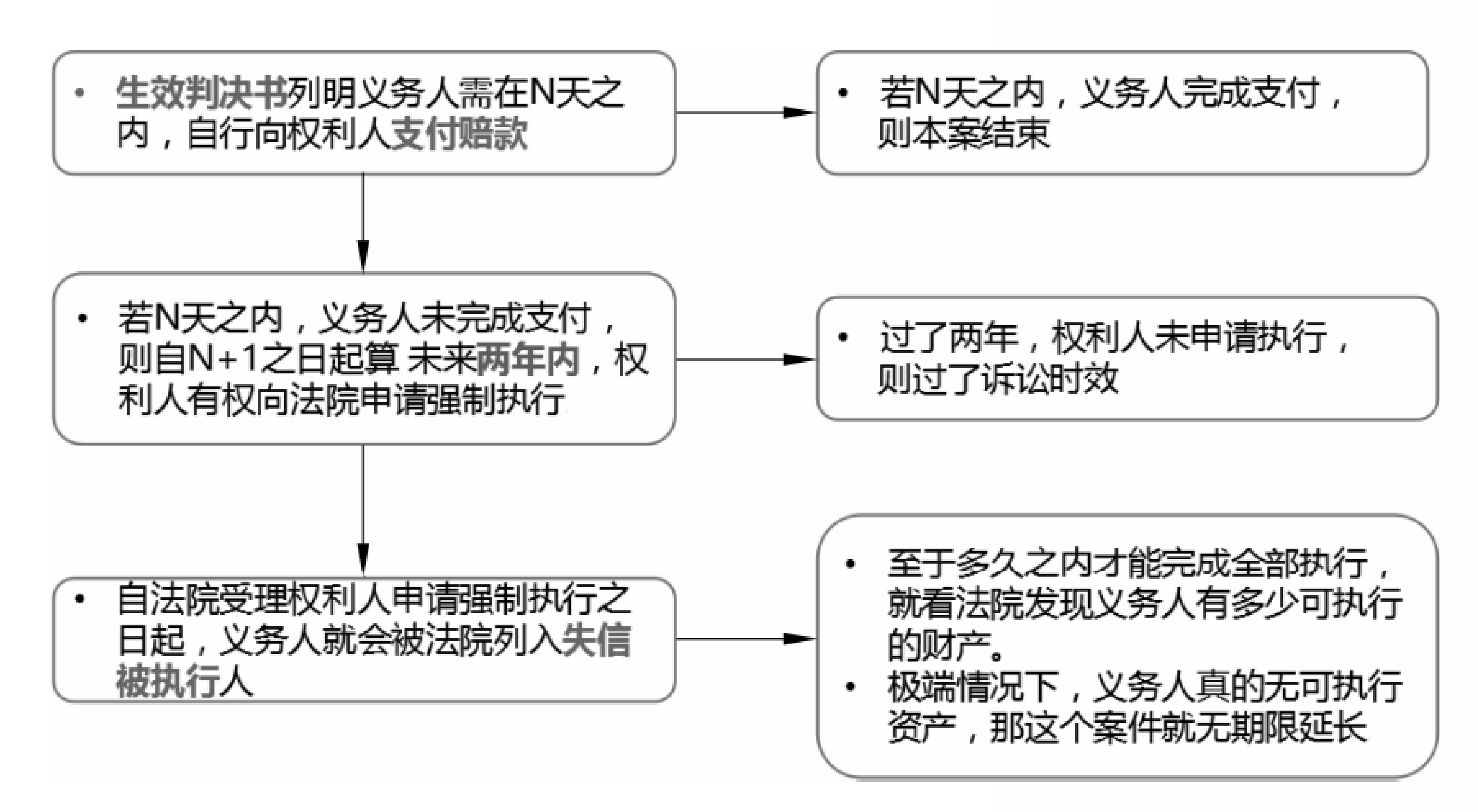


图 5.13　涉诉且赔款流程

除了司法相关的信息可用于提前判断企业的财务状况外，笔者研究发现存量债券到期收益率的异常波动，也是企业还款能力变差的一个明显信号，且该信号基本可在违约前 6 个月以上出现，显然比较有价值。通过研究已经违约的 164 只债券，发现基本都在违约前 6 个月以上出现到期收益率异常的情况，如图 5.14 所示。出现这种信号的主要原因是在企业违约前，基本都会存在“市场灵通人士”提前获取这家公司现金流紧张、还款能力变差的信息，从而在市场上低价抛售这家公司的债券。

除了上述场外定性数据以外，笔者还提取了如下的定性入模指标。

(1) 最近 1 年被诉且需要赔款的次数占该主体所有同类案件的比重。

(2) 最近 2 年被诉且需要赔款的次数占该主体所有同类案件的比重。

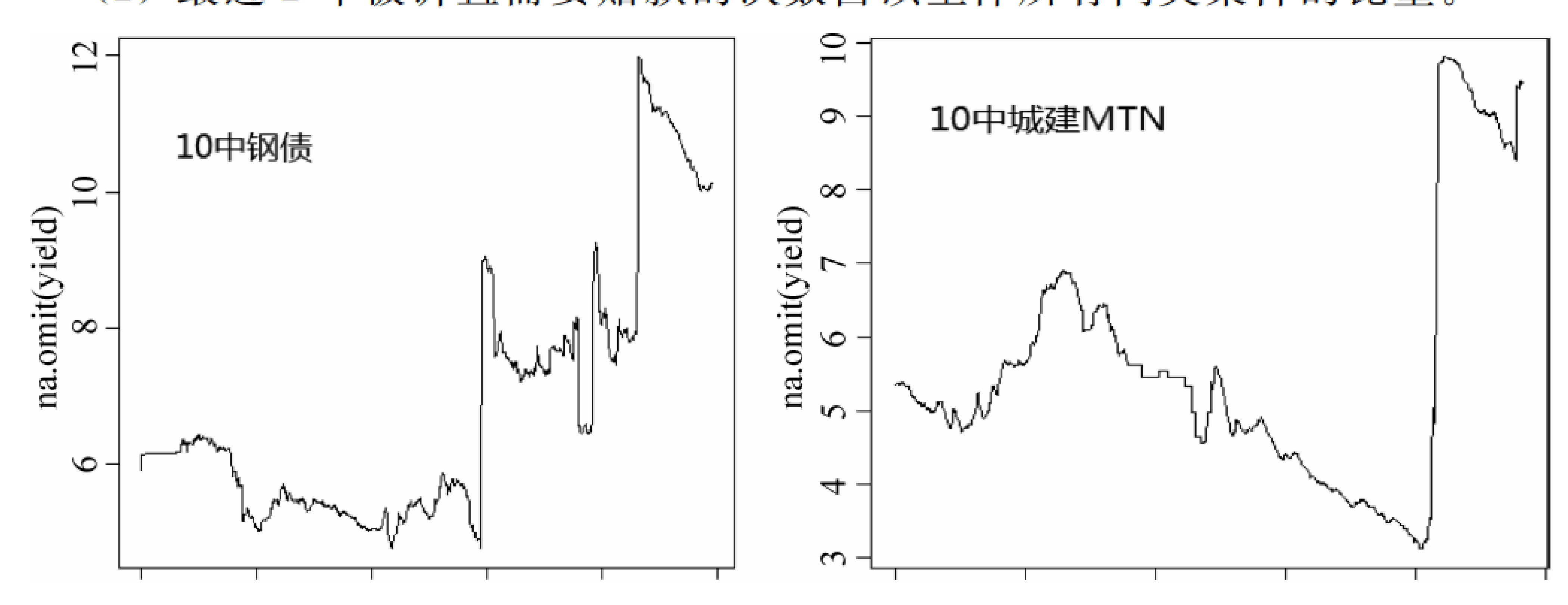


图 5.14　部分违约债券违约前到期收益率的变化



图 5.14 （续）

（3）最近半年被执行次数占该主体被执行总数的比。

（4）最近半年失信被执行次数占该主体被失信执行总数的比。

（5）近 2 年招聘变化趋势。

（6）是否存在被银行起诉。

这些数据都是无法粉饰和造假的。笔者使用上述指标并使用实质性违约样本作为违约标记，通过分别获取 2013 年、2014 年和 2015 年的历史数据，开发模型并预测下一年的违约分类情况，取得了很好的结果。此处，仅分析测试结果，详细的 Python 代码和模型开发过程将在接下来的章节中详述。

2014 年全年共计有 6 只债券违约，其中 4 只民营企业债券和 2 只外资企业债券。按资金募集方式分类，其中 5 只为私募债（无评级）、1 只为公司债（有评级）。并且 2014 年发生的债券违约事件均为利息违约（因为债券到期日均在 2015 年以后）。通过获取 2014 年以前的数据开发模型，如果假定违约概率大于 30％内部判定为违约，则模型的准确率为 83.33％，如图 5.15 所示。2014 年的债券违约出现了两个特殊情况，分别是：①13 华珠债，为私募债，违约前无任何异常信息披露（司法、财务等）；②13 中森债，为私募债，违约前 1 个月内，有异常

司法信息披露，但时间太短，基本无法交易。

用同样的方法，我们获取 2015 年以前的数据，开发模型并预测 2015 年债券违约的情况，如图 5.16 所示。2015 年全年共计 23 只债券违约，其中 15 只民营企业债券、3 只外资企业债券、5 只国有企业债券，这些债券共计 13 只私募债。与 2014 年相比，由于 2015 年违约的私募债占比减少，信息披露的完整性变好，如果仍以大于 30％的违约概率判定为违约，则预测准确率提升到 91.30％。

2015 年的债券违约事件，也出现了特殊情况，那就是中央国有企业（天威集团）第一次出现违约，且该主体共计 4 只债券发生违约。根据主体性质的不同，我们通常将违约原因分为两类：①个人主体发生违约，通常是“意愿”的问题。因为人的随意性很强，时常可能出现有钱但忘记或不愿还款的情况。②机构主体发生违约，通常是“能力”的问题。因为机构的主观性不强，且机构一旦发生违约或负面信息，会对公司的声誉造成很大影响，且会影响公司的正常经营。因此，机构主体发生的违约通常真的是由其还款能力不足造成的。但 2015 年出现的天威集团违约事件或今后可能出现的其他央企或国企子公司、城投债违约事件，大概率是央企或国企不愿意还款的“意愿”问题。因为在中国特色的社会主义国情下，央企或国企的还款能力是不容置疑的。

2016 年全年共计 78 只债券违约，包括场内（银行间和沪深交易所）56 只和场外（区域股权交易中心）22 只；由于 2016 年非私募债违约占比增多且信息披露逐渐完善，模型的预测准确率也不断提升，如图 5.17 所示。

通过对上述测试结果的分析可知，由于私募债发债主体的信息披露非常不完整、不及时，造成关键数据的缺失，影响了判断准确率。在我们剔除私募债后，最近 3 年的回测结果均有明显提升，如图 5.18 所示。这个测试结果给了我们很大的启示，那就是信息披露历史不完整的债券，不投资。

2014 年，只有 1 只非私募债券违约，且是上市公司，信息披露较完善；违约概率阈值取 30％时，100％地识别出了违约事件。

2015 年，有 9 只非私募债券违约，信息披露较完善；违约概率阈值取 50％时，有 88.89％（8 只）的债券准确识别出了违约事件。只有 14 波鸿 CP001 误判，主要由于关键信息（借款未还被起诉）滞后造成。

2016 年，有 47 只非私募债券违约，信息披露较完善；违约概率阈值取 30％时，有 97.87％（45 只）的债券识别出了违约事件。有 14 益优 02、15 华昱 CP001 2 只债券由于信息披露不完整、不及时被误判。

第 6 章 资本市场信用债投资分析的中国特色

本章将重点介绍中国资本市场信用债投资分析时，面临的现状及解决方案。在现阶段，信用债投资分析面临财务数据存在一定的粉饰现象、对违约前的征兆预测不显著等客观问题。本章将给出克服这些客观问题的解决方案，并给出财务粉饰的本福特法则统计识别方法。

本章共分为四节，6.1 节重点介绍 MySQL 数据库配置和自动抓取建模所需数据的方法，6.2 节重点介绍使用统计方法检验财务数据对违约状态的影响是否显著，6.3 节重点介绍使用统计方法检验非财务的场外数据对违约状况的影响是否显著，6.4 节重点介绍财务粉饰的本福特统计检验方法。

6.1 数据库配置和数据抓取的 Python 源代码

本节以开源、免费的数据库软件 MySQL 为例，逐步讲述其安装和配置过程，这也是建设信用债投资分析所需的信用风险数据库的必要前提。各步骤的详细过程如下所示。

（1）访问 MySQL 首页（http://www.mysql.com/），如图 6.1 所示。

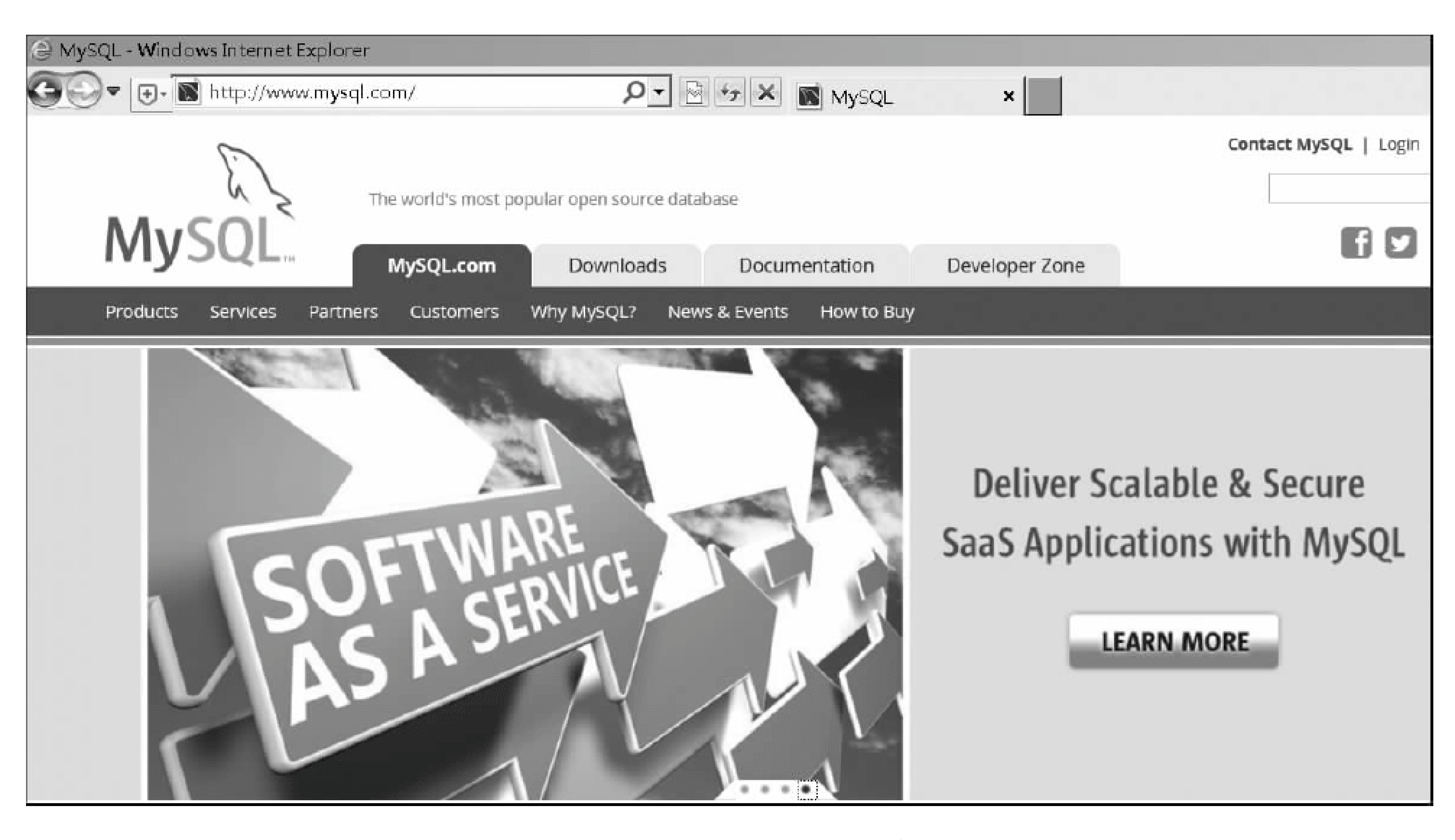


图 6.1　访问 MySQL 首页

(2) 下载免费的 MySQL 社区版，选择免费的社区版 MySQL 服务器(Community Server)，单击“Downloads”，如图 6.2 所示。



图 6.2 下载 MySQL Community 版本

(3) 选择 Windows 平台的版本，在“Select Platform”下拉列表中选择“Microsoft Windows”平台，并根据自己电脑的配置选择 32 位或 64 位平台(本书以 32 位机器为例)，单击“Download”下载，如图 6.3 所示。

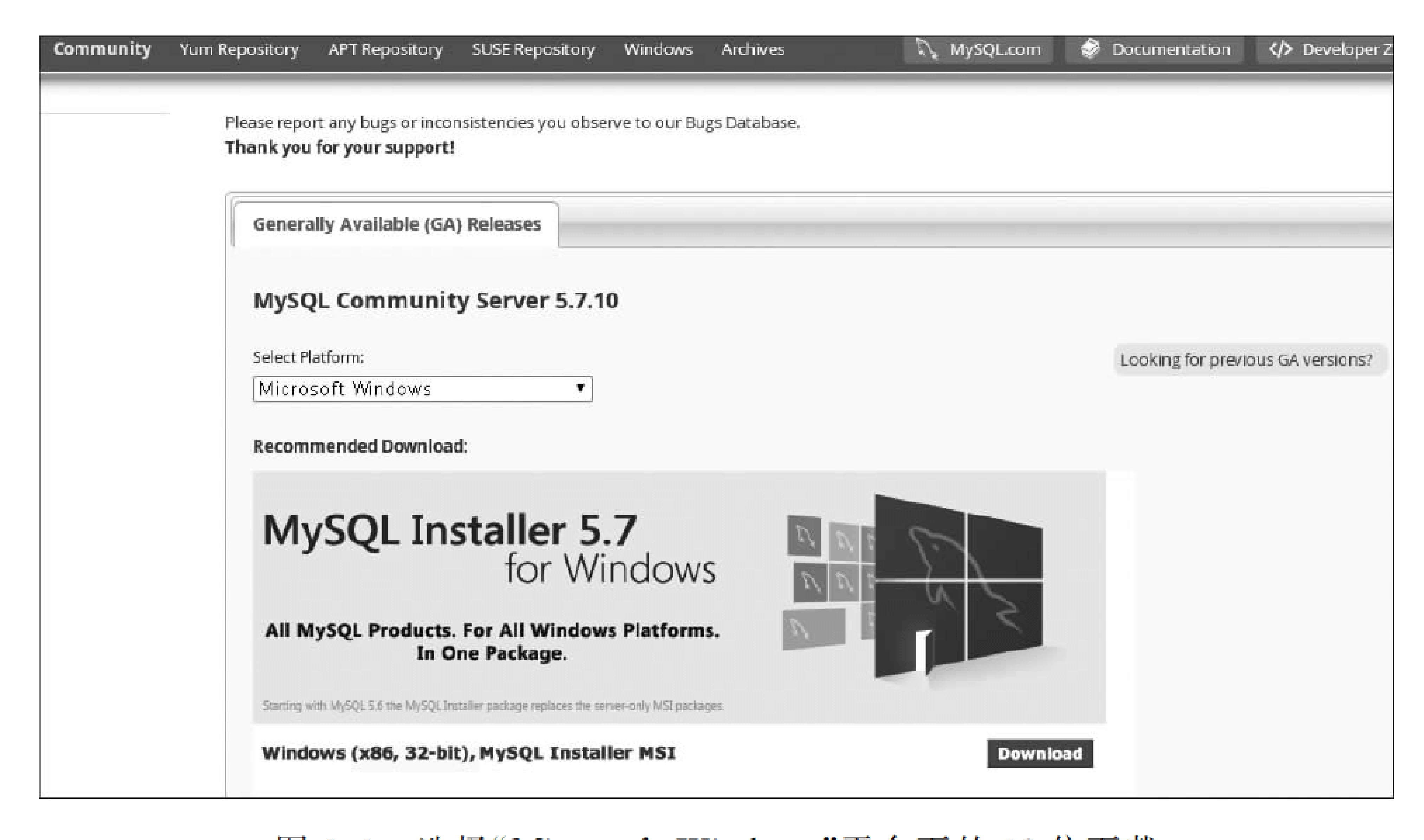


图 6.3 选择“Microsoft Windows”平台下的 32 位下载

(4) 安装并配置 MySQL Community Server，下载完成后，双击安装，弹出图 6.4 所示的安装界面。

(5) 勾选“I accept the license terms”，单击“Next”按钮，选择数据库的安装类型，如图 6.5 所示。

(6) 选择“Developer Default”，单击“Next”按钮，检查安装 MySQL 所需要

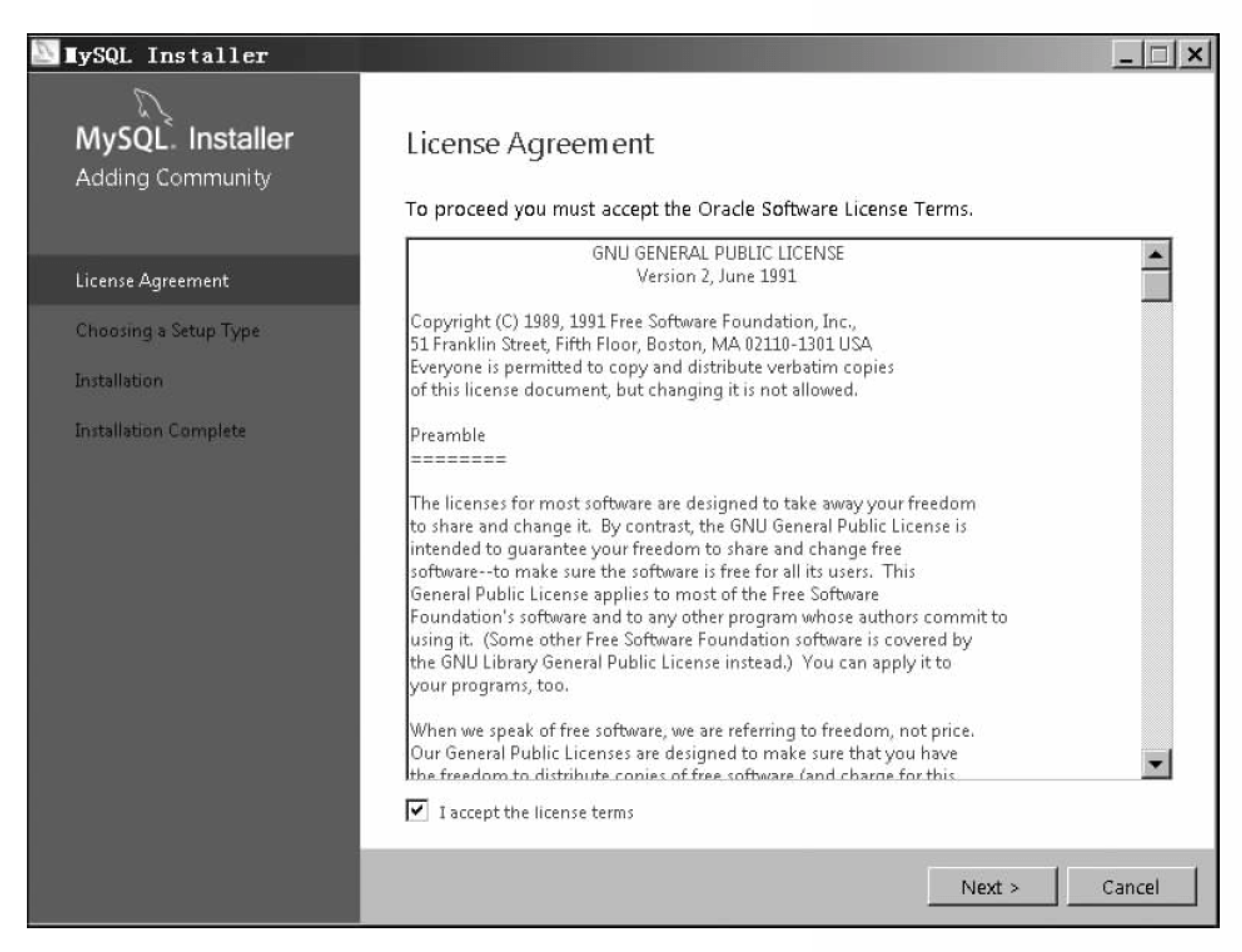


图 6.4　MySQL 安装界面

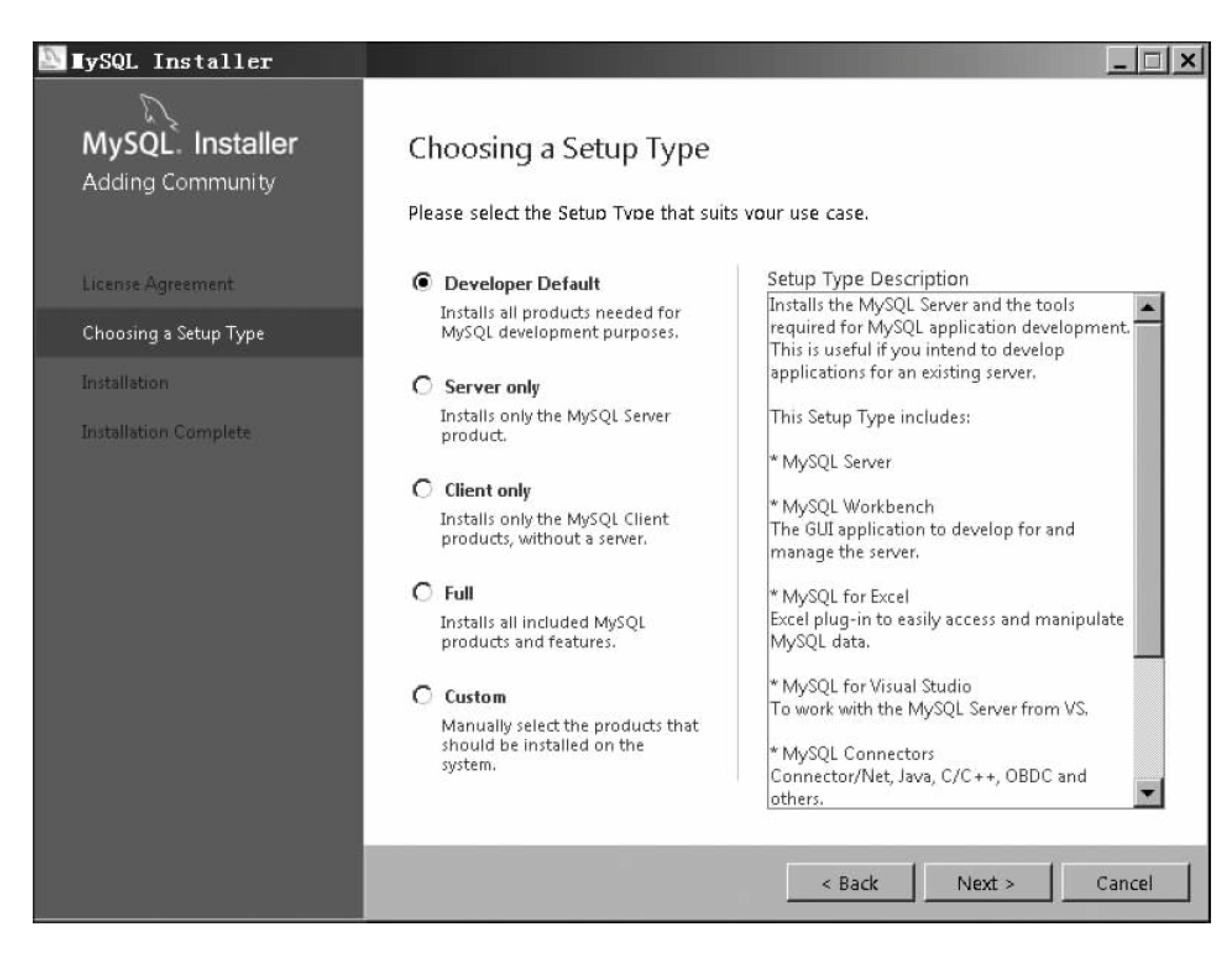


图 6.5　选择数据库的安装类型

的环境配置，如图 6.6 所示。

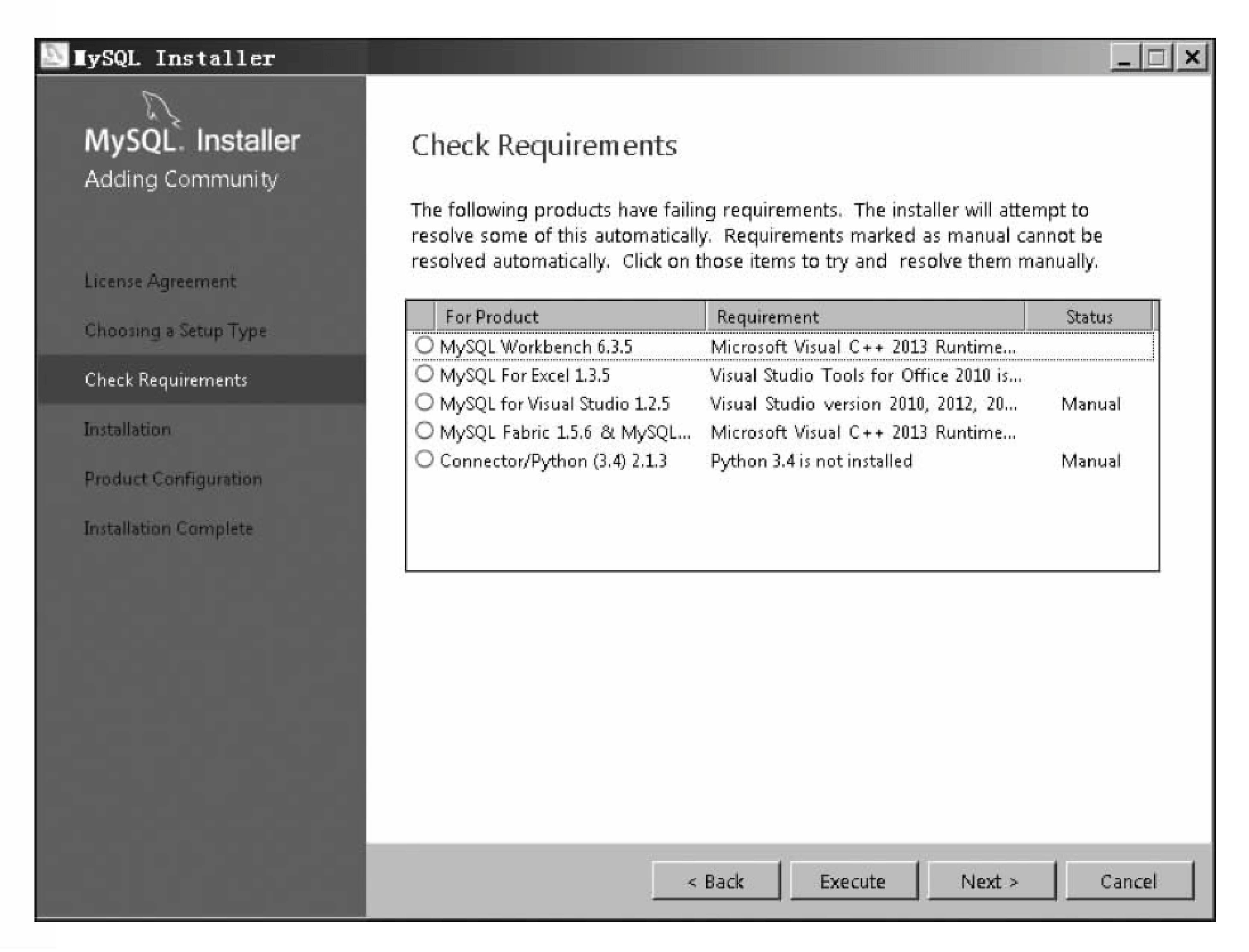


图 6.6　检查安装 MySQL 所需要的环境配置

（7）单击图 6.6 所示的“Execute”按钮，环境配置检查结果如图 6.7 所示。

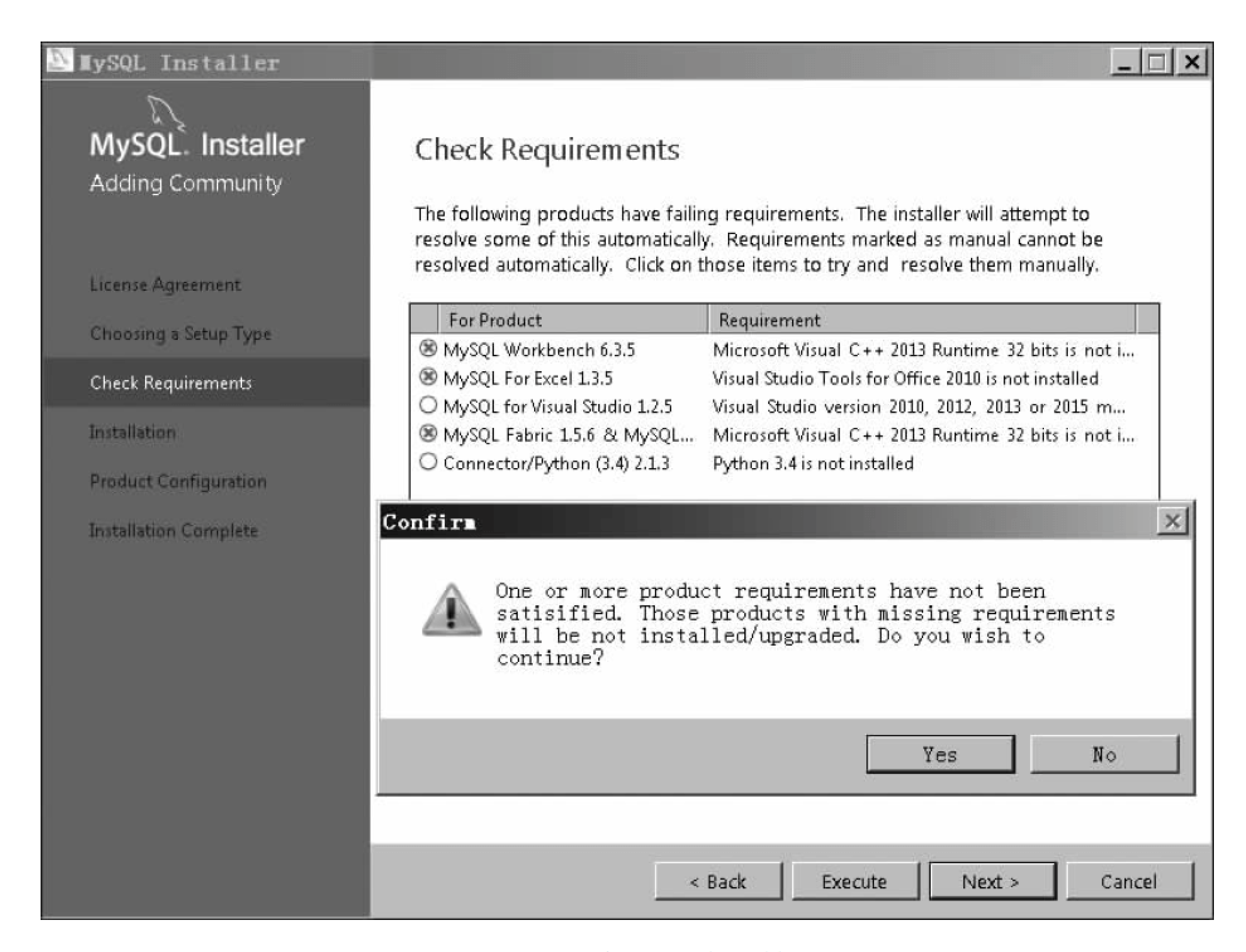


图 6.7　环境配置检查结果

（8）弹出“Confirm”对话框，如图 6.7 所示。单击“Yes”，然后单击“Next”按钮，弹出准备安装对话框，如图 6.8 所示。

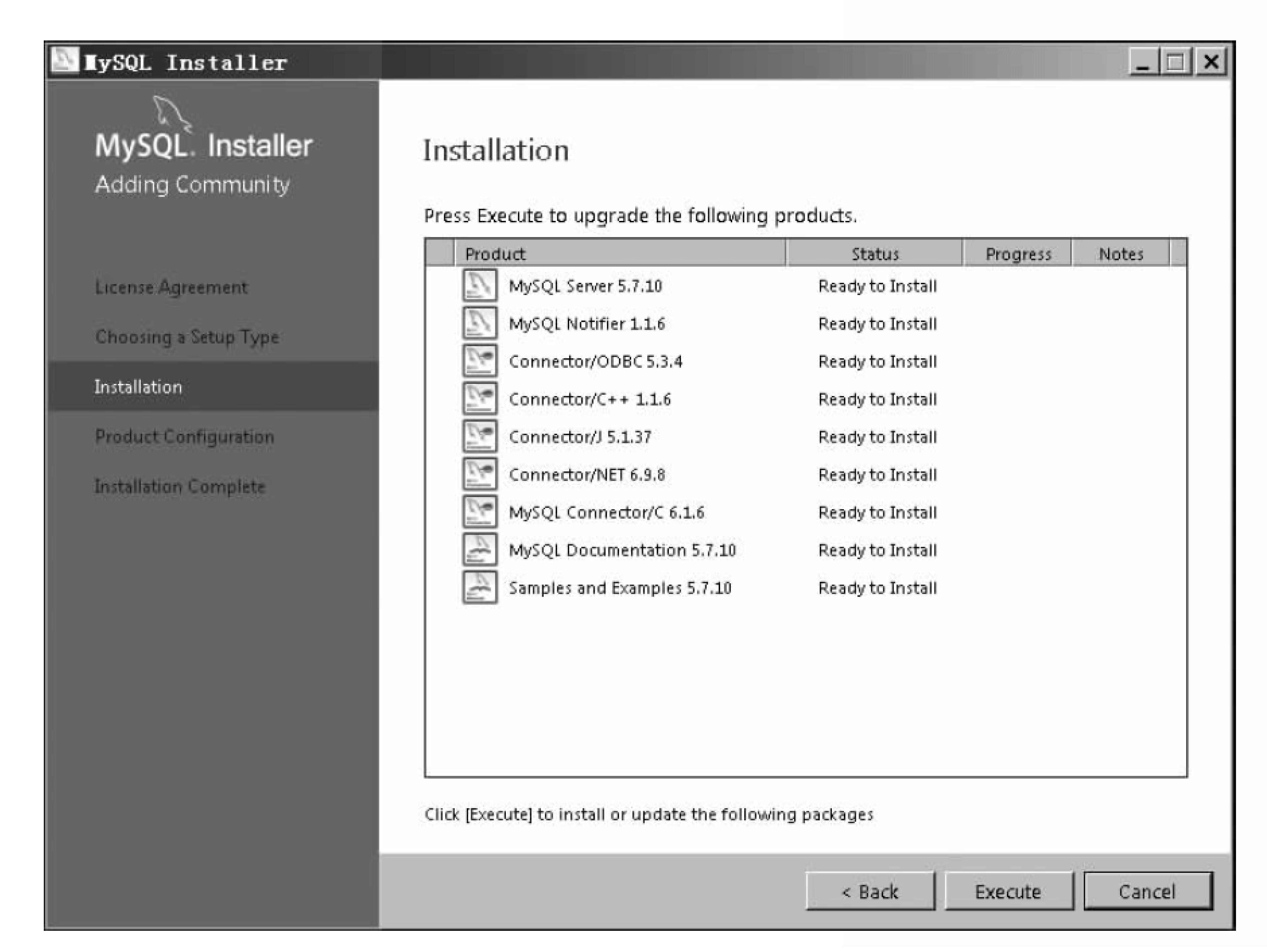


图 6.8 准备安装对话框

（9）单击图 6.8 所示对话框中的“Execute”按钮，等待数据库安装。安装结束后，会显示图 6.9 所示的对话框。

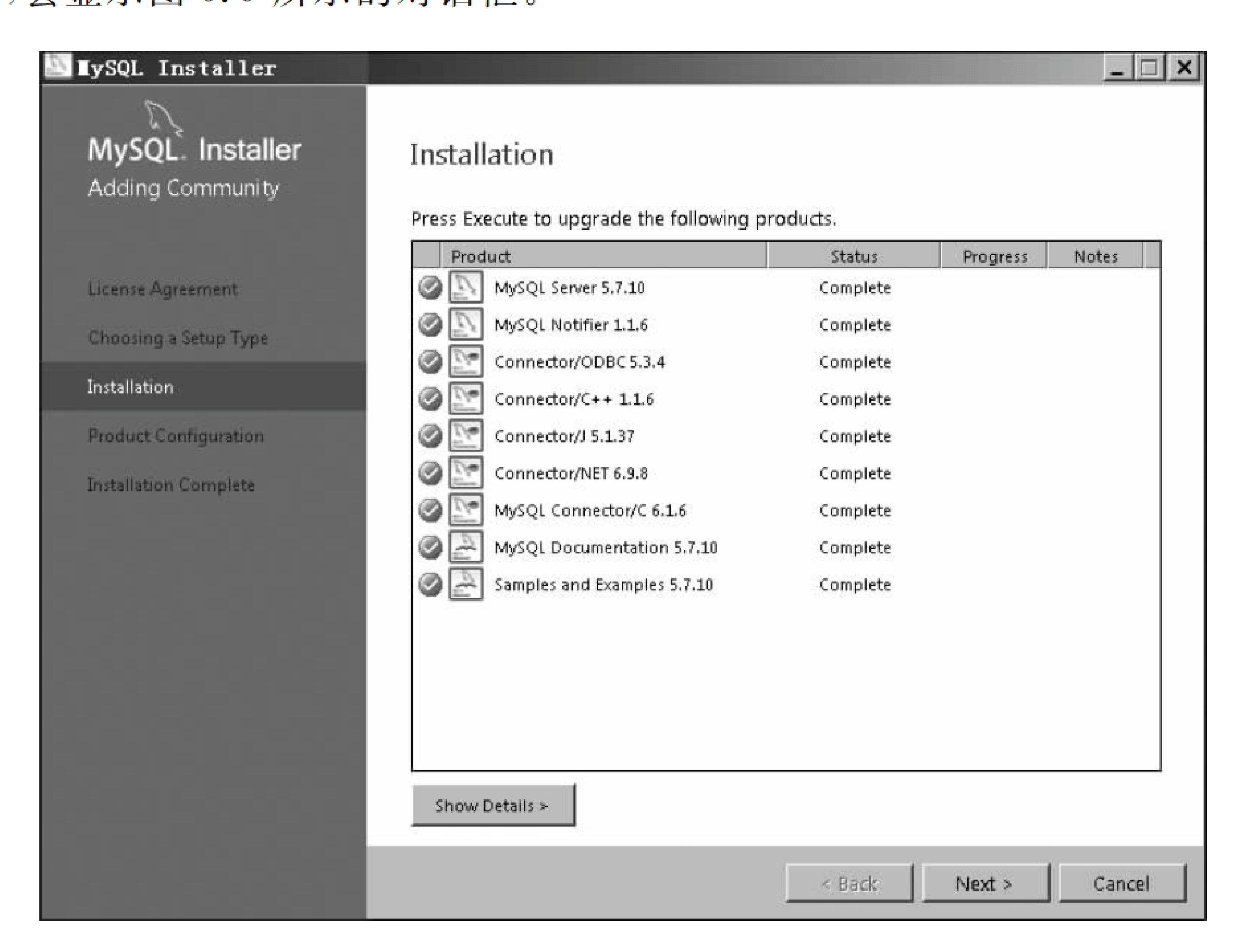


图 6.9 数据库安装完成对话框

(10) 单击图 6.9 所示对话框中的“Next”按钮，进入数据库配置对话框，如图 6.10 所示。

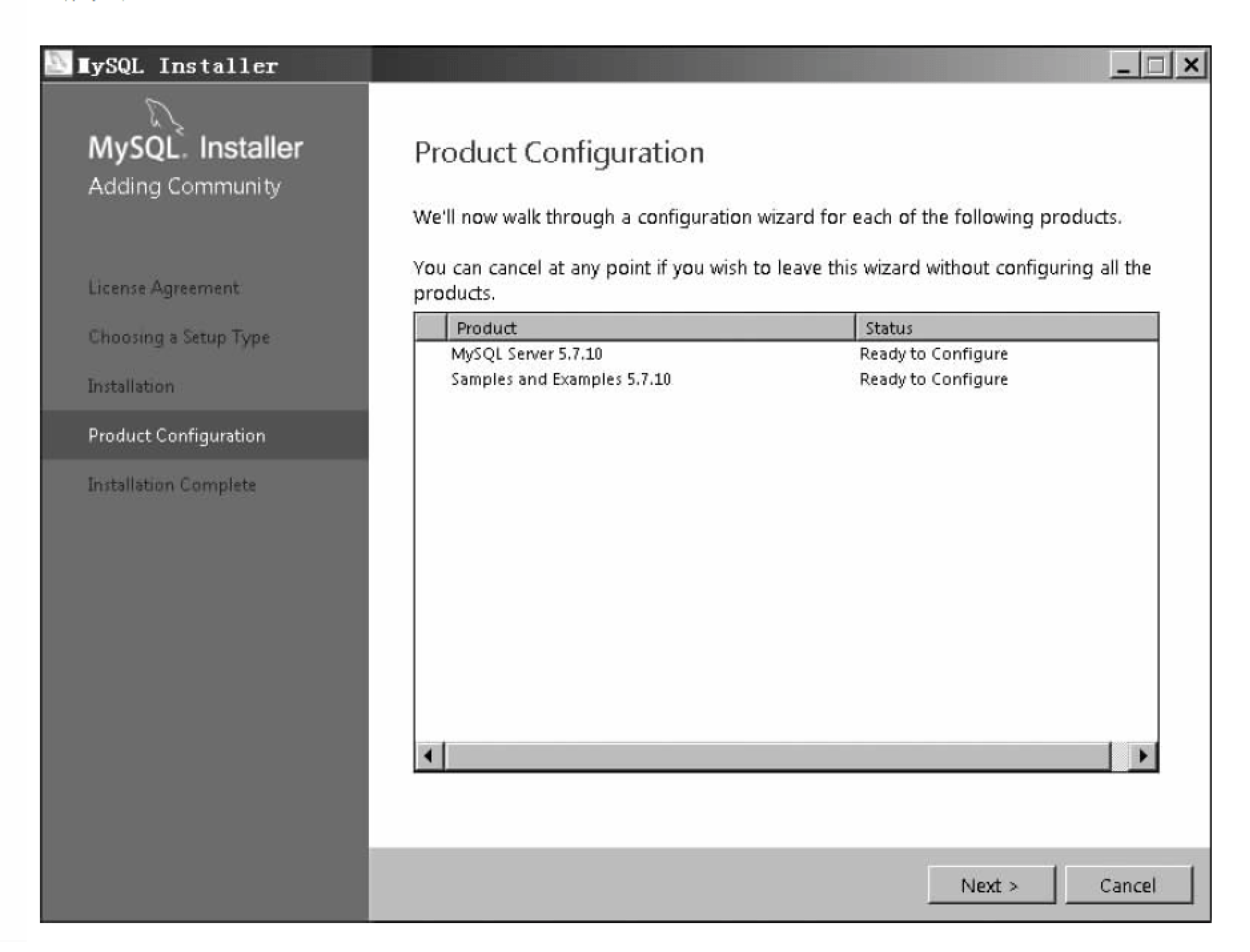


图 6.10　数据库配置对话框

(11) 选择数据库服务器的配置类型，如图 6.11 所示。有三种类型可选，作为示例，本书选择“Development Machine”，读者可根据实际情况选择其他类型。

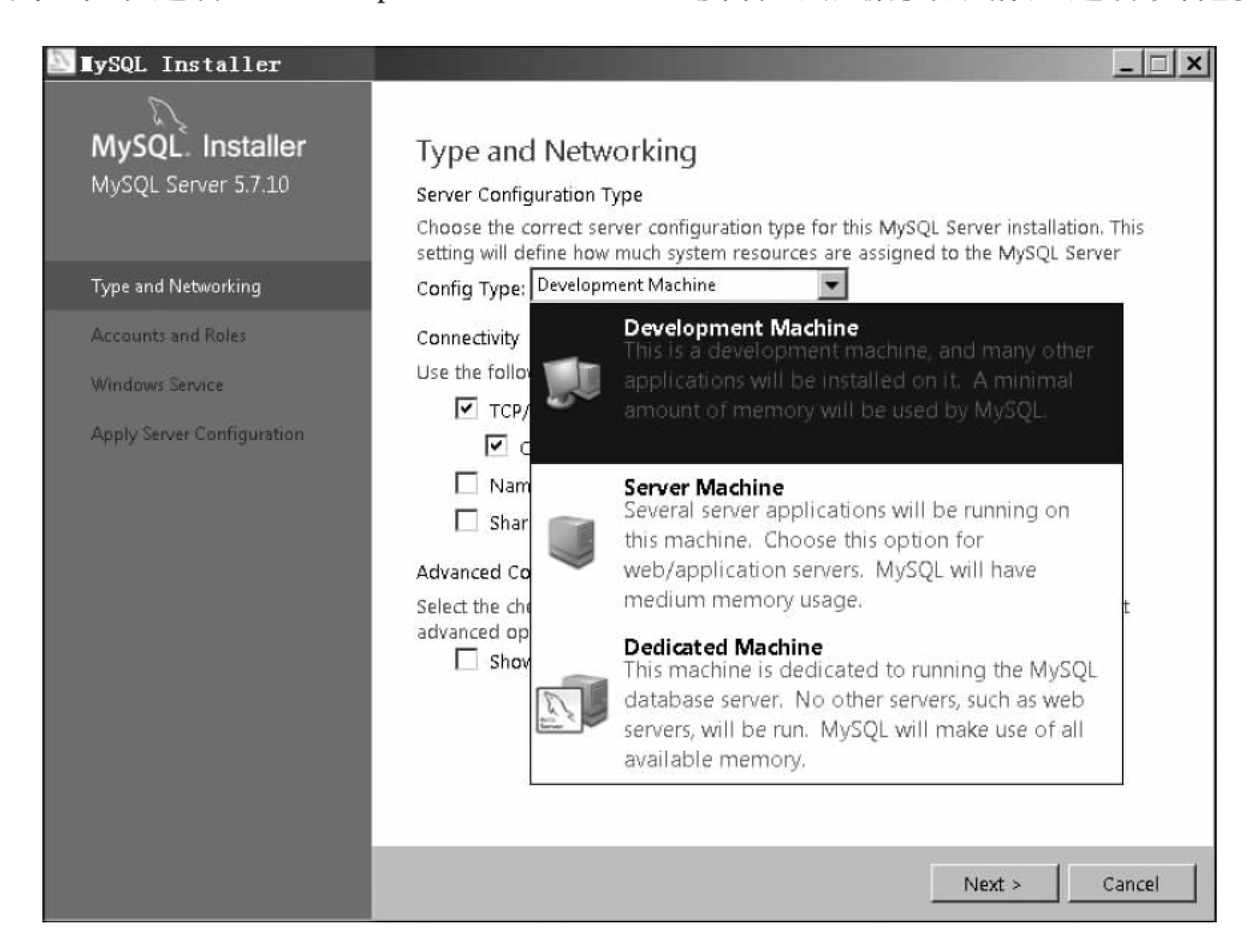


图 6.11　选择数据库服务器的配置类型

（12）单击图6.11中的“Next”按钮，并设置根用户（root）密码，即为超级用户密码（本书设置为admin），如图6.12所示。

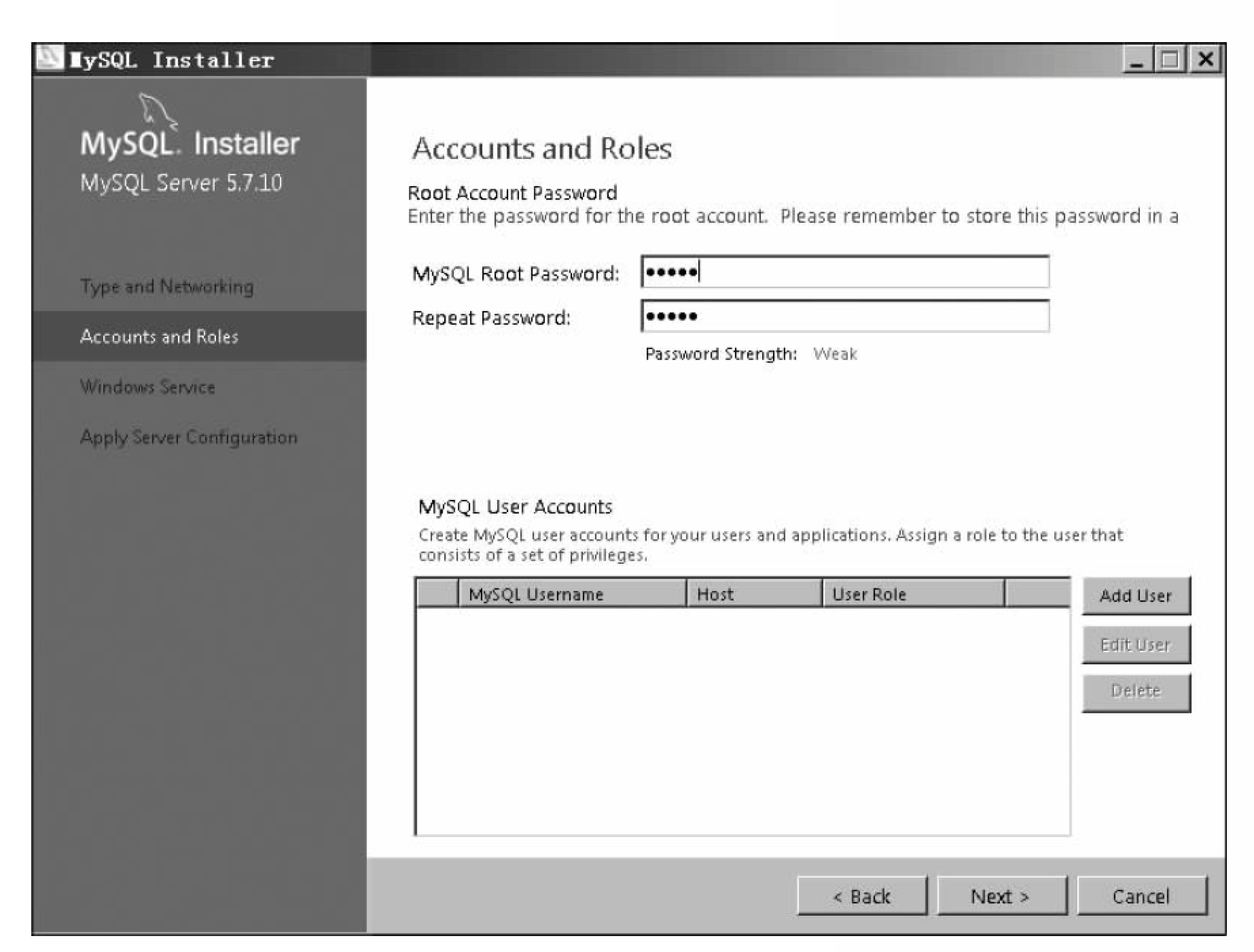


图6.12 设置根用户密码

（13）通过单击图6.12中的“Add User”按钮，添加新用户，本书添加新用户“test”，密码为admin，如图6.13所示。

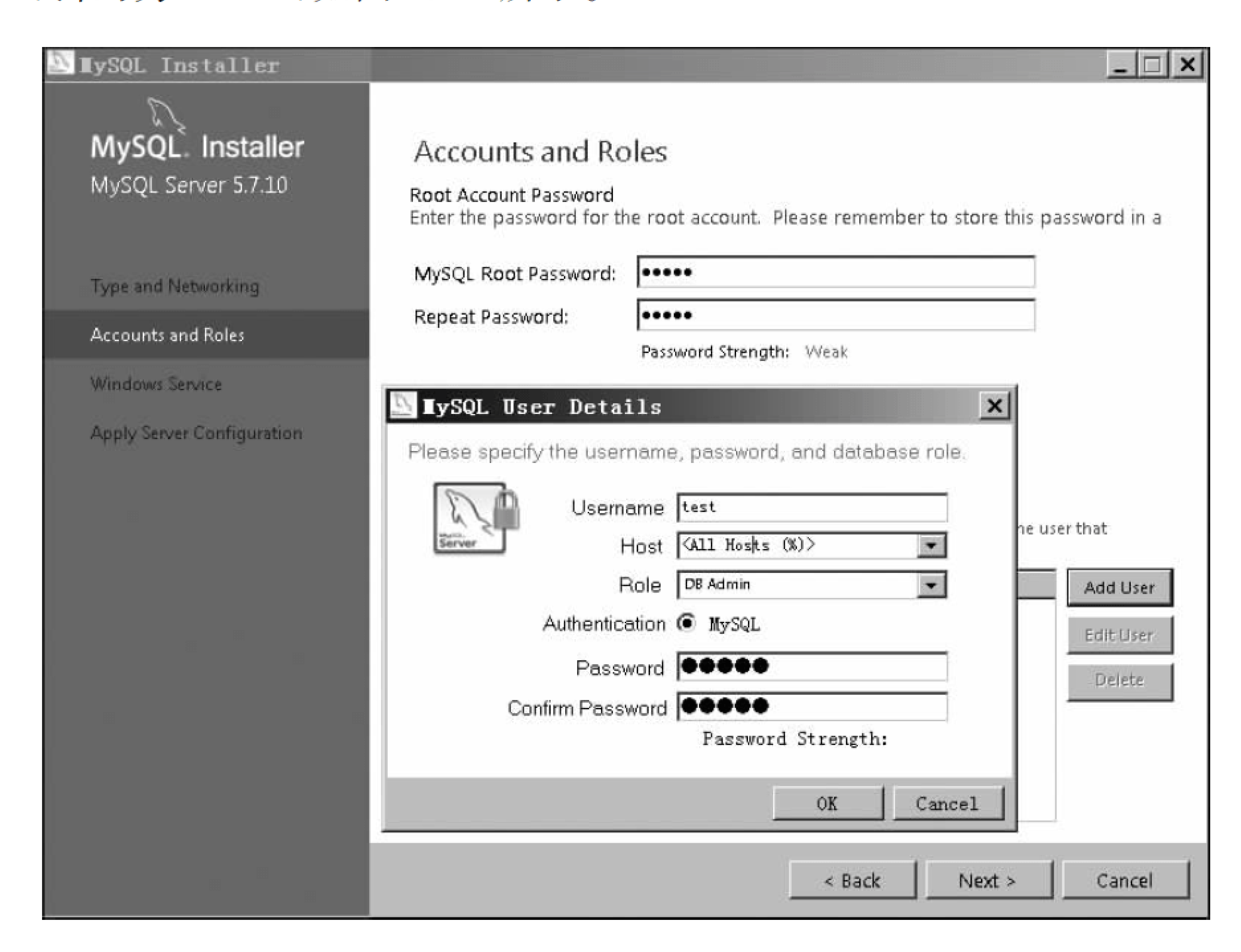


图6.13 添加新用户并设置密码

(14) 为新添加的用户 test 设置“Host”和“Role”,本书中“Host”设置为“localhost”,“Role”设置为“DB Admin”,如图 6.14 所示。

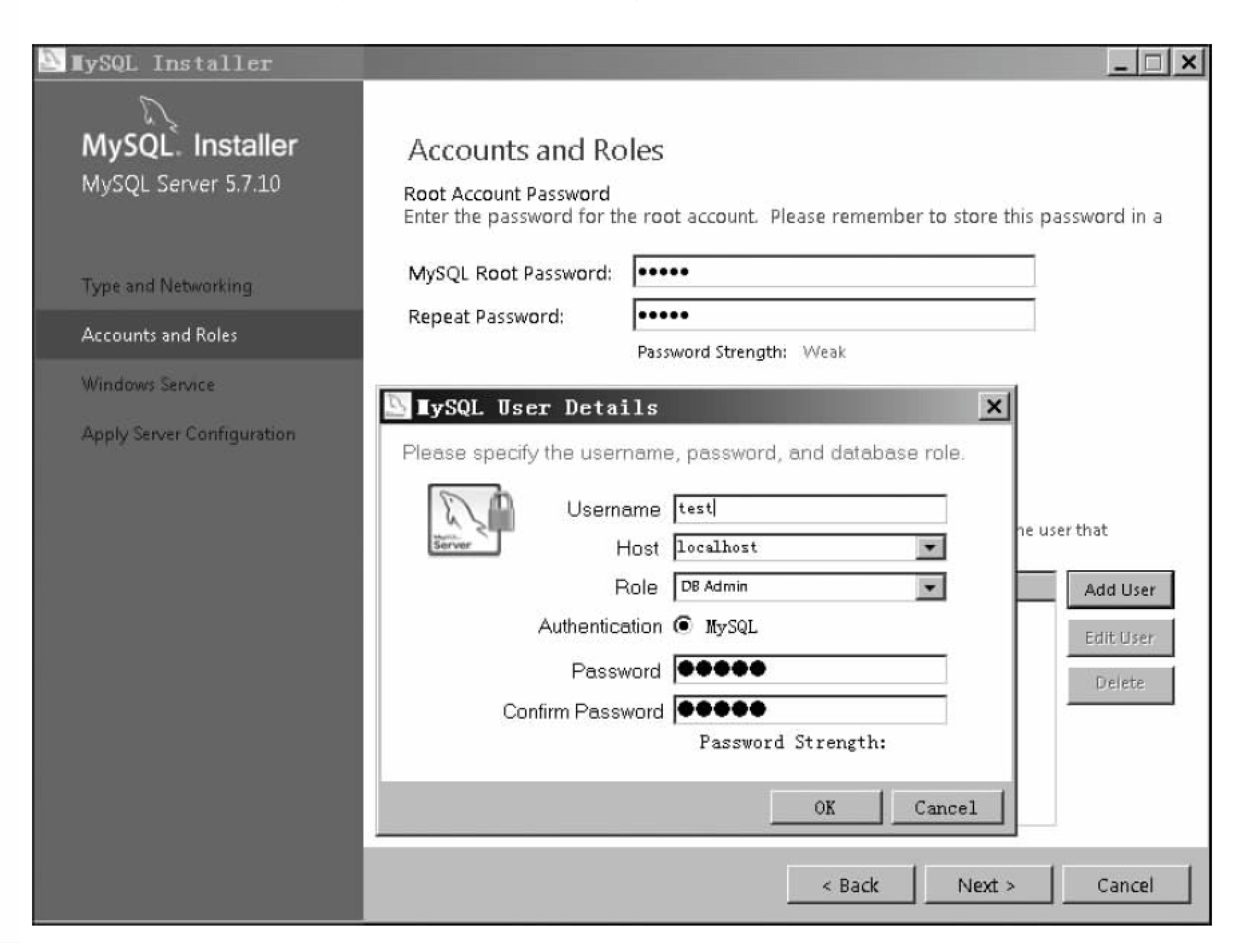


图 6.14　为新用户 test 设置“Host”和“Role”

(15) 单击图 6.14 中的“OK”按钮,完成新用户添加,并单击“Next”按钮,设置 Windows 服务的名称,如图 6.15 所示。

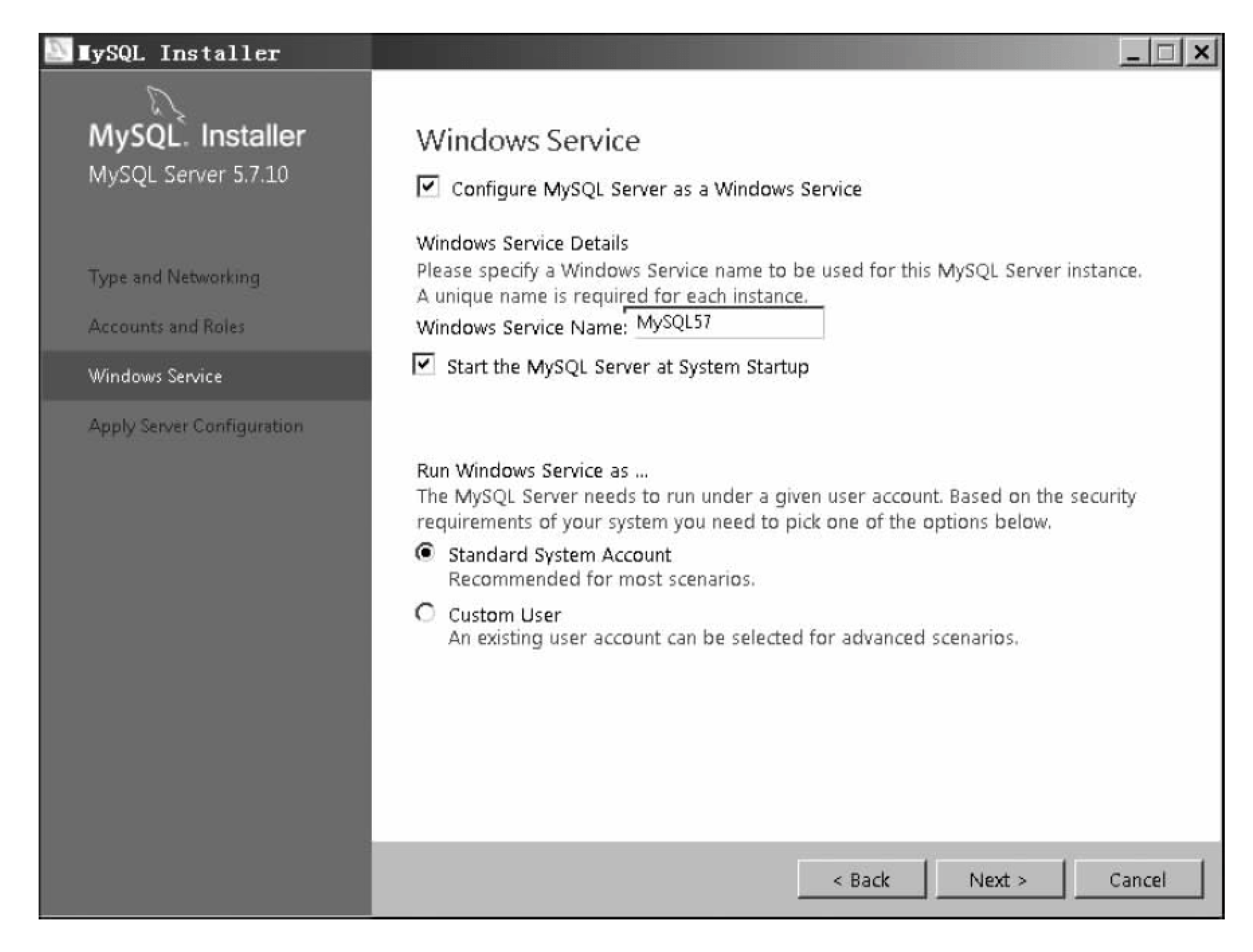


图 6.15　设置 Windows 服务的名称

（16）本书直接采用默认的 Windows 服务名称“MySQL57”，单击图 6.15 中的“Next”按钮，进入数据库配置确认对话框，如图 6.16 所示。

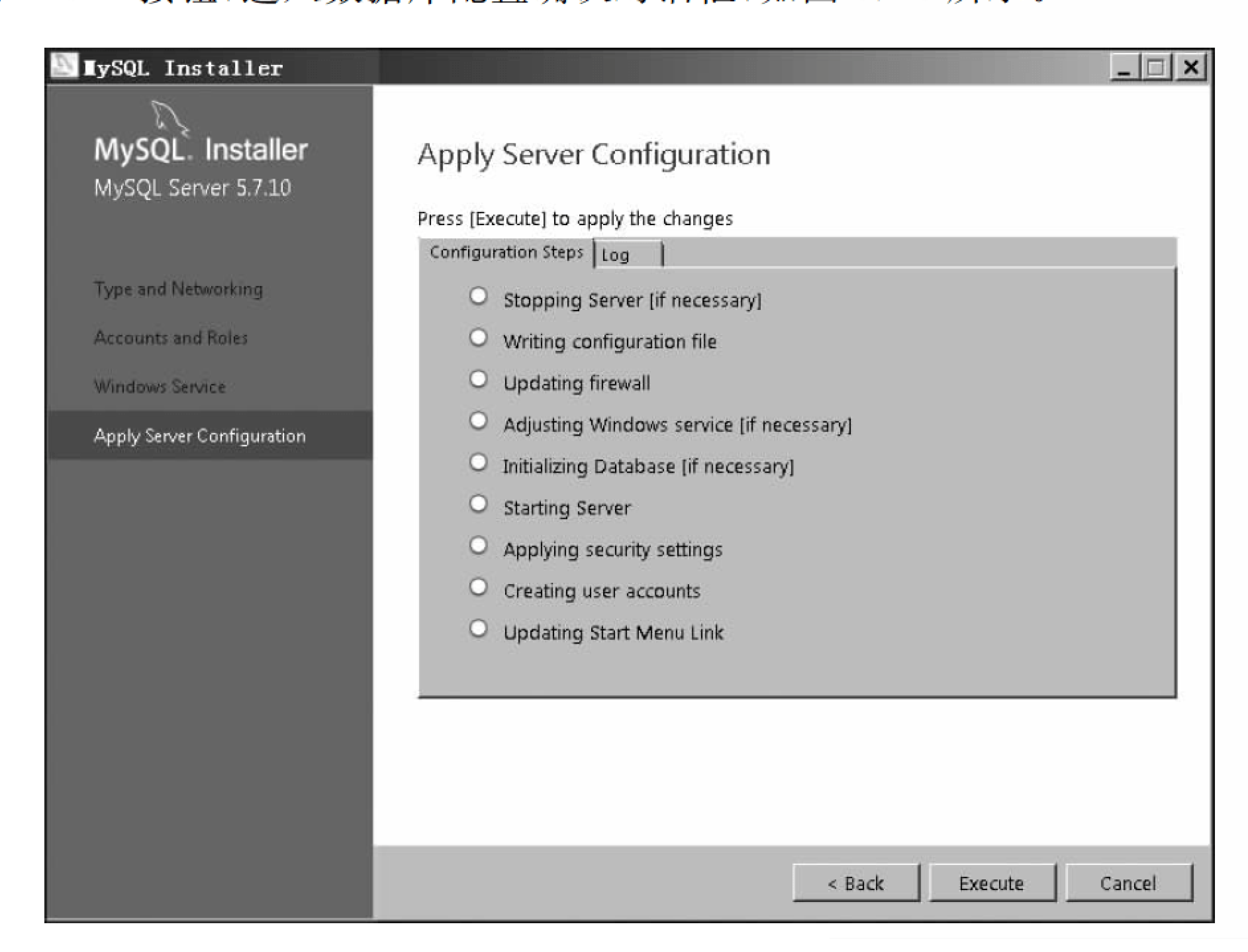


图 6.16　数据库配置确认对话框

（17）单击图 6.16 中的“Execute”按钮，使之前的设置生效。最后，单击图 6.17 所示对话框中的“Finish”按钮，完成数据库的配置。

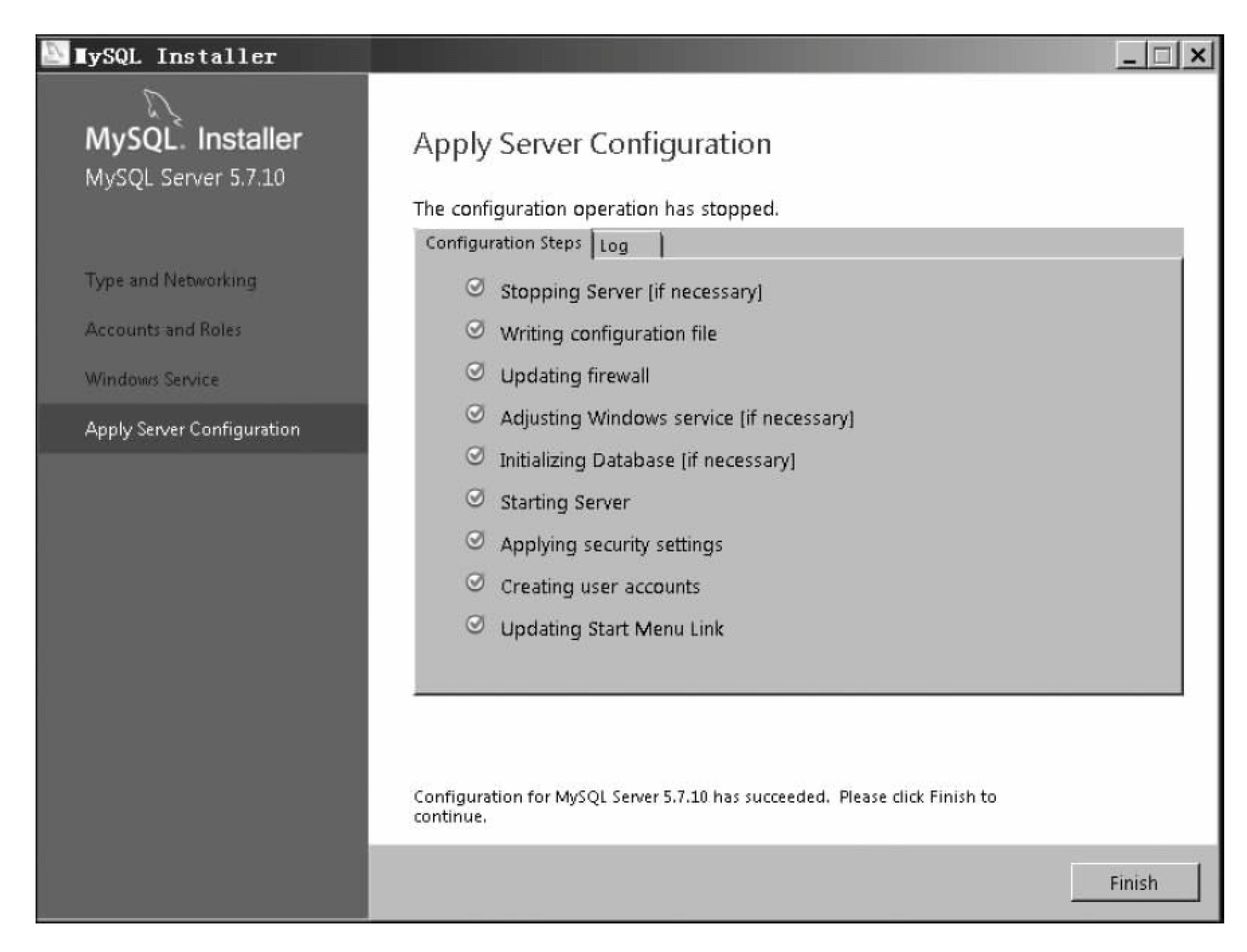


图 6.17　完成数据库配置

至此，MySQL 数据库安装、配置完毕。

(18) 建立存储数据的数据库，并进行远程访问授权。经过前面 17 个步骤的安装与配置，我们搭建好了“房子”的框架。接下来，我们“装修该房子”，并分出一个具体的“房间”(数据库)来存储建模所需的数据。

回到电脑桌面，单击“开始”→“所有程序”→“MySQL”→“MySQL Server 5.7”→“MySQL5.7 Command Line Client”，如图 6.18 所示。

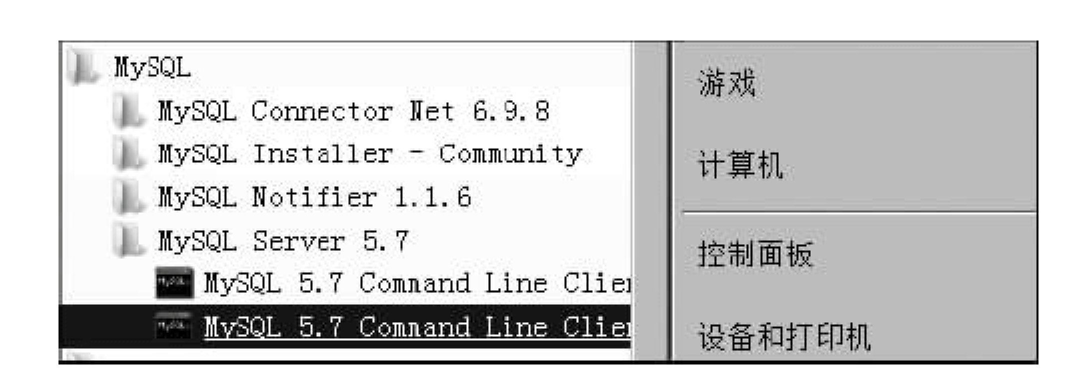


图 6.18 启动 MySQL 脚本

弹出图 6.19 所示的对话框，输入数据库配置阶段设置的密码：admin

```
MySQL 5.7 Command Line Client
Enter password: _
```

图 6.19 输入数据库配置阶段设置的密码

显示图 6.20 所示内容，表明数据库连接成功。

```
MySQL 5.7 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.7.10-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

图 6.20 链接 MySQL

这样，我们就以超级用户身份登录了 MySQL 数据库服务器。下一步是建立存储建模所需数据的数据库，我们命名为 creditrisk。在图 6.20 中的光标后，输入“create database creditrisk;”，并按回车键，返回图 6.21 所示信息，表明数据库建设完成。这样，我们就建立了专门存放模型开发所需数据的数据库，数据库名称为“creditrisk”。

接下来我们为远程访问该数据库进行授权，假设需要远程访问该数据的机

```
MySQL 5.7 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.7.10-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database creditrisk;
Query OK, 1 row affected (0.00 sec)

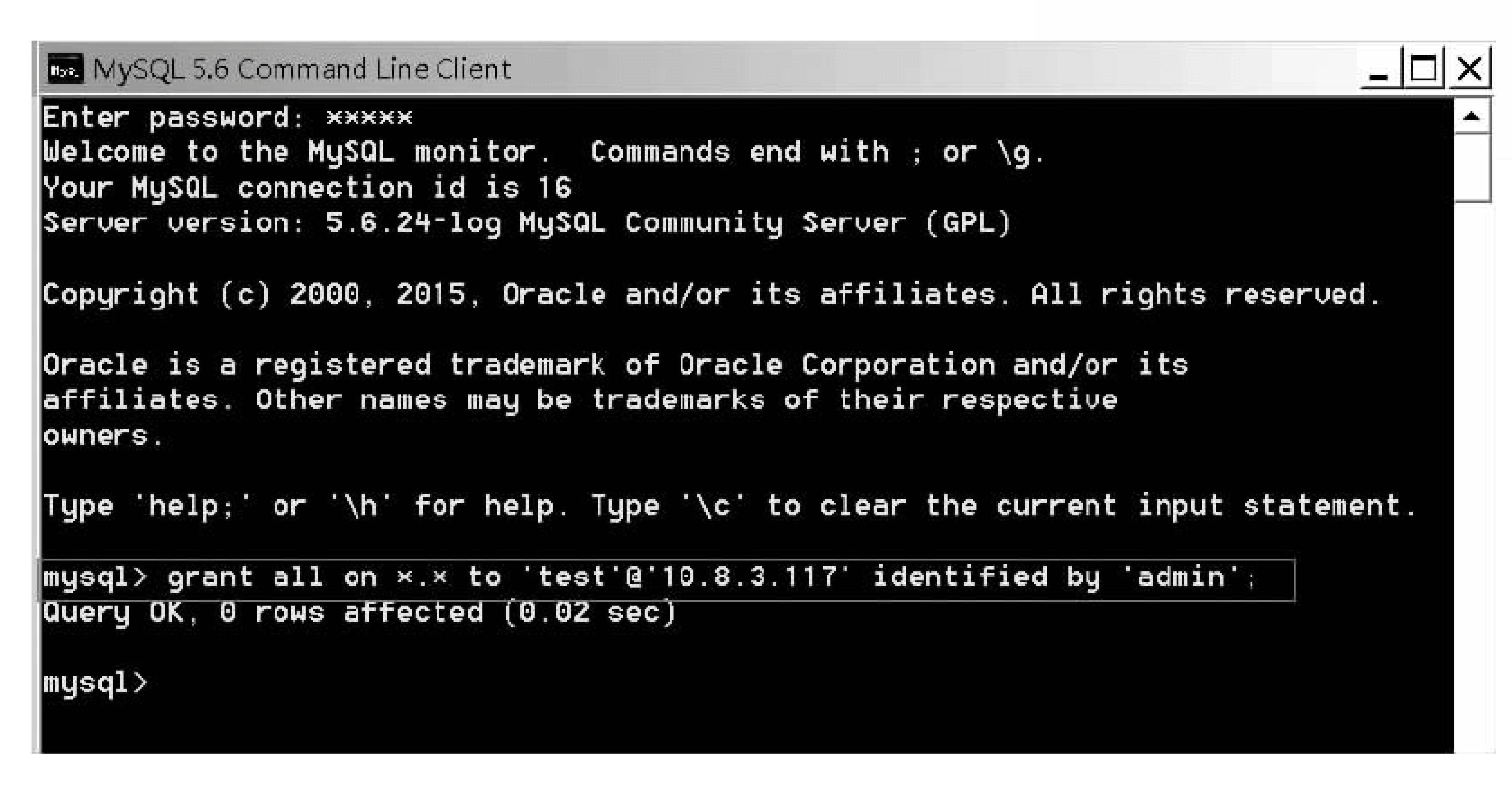
mysql>
```

图 6.21　建立“creditrisk”数据库

器 IP 地址为“10.8.3.117”,则可用如下 SQL 脚本实现授权:

```
grant all on * . * to 'test' @'10.8.3.117' identified by 'admin';
```

该句 SQL 脚本的意思是授权“10.8.3.117”这台机器,以用户名“test”、密码“admin”访问数据库,返回如图 6.22 所示的内容,表明授权成功。



```
MySQL 5.6 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 5.6.24-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> grant all on *.* to 'test'@'10.8.3.117' identified by 'admin';
Query OK, 0 rows affected (0.02 sec)

mysql>
```

图 6.22　远程访问授权

在图 6.22 中输入命令:quit,并按回车键退出 MySQL 服务器。

至此,数据库“creditrisk”和远程访问授权均建设完毕。

(19) 获取安装 MySQL Server 机器的网址,方法如下:

回到 Windows 桌面,单击“开始”→“输入 cmd”→“回车”,如图 6.23 所示。

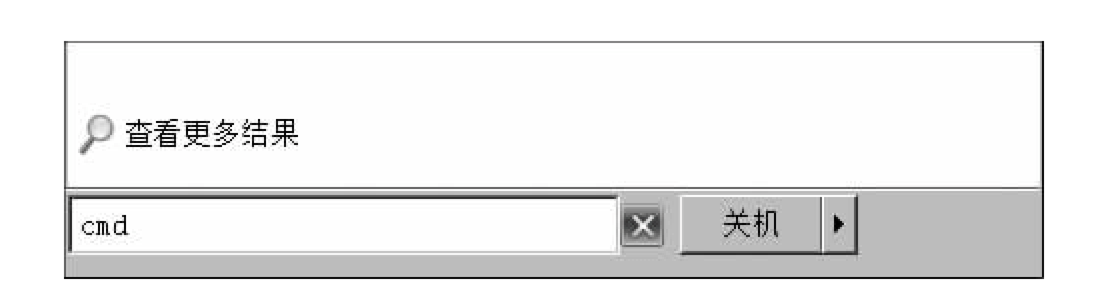


图 6.23　打开命令行对话框

弹出图 6.24 所示对话框,并输入“ipconfig”,按回车键,获取数据库服务器

的 IP 地址。图 6.24 的输出结果表明,IP 地址为“10.8.16.210”。



图 6.24　获取本机 IP 地址对话框

建立远程连接时,可在安装 MySQL Server 的机器上,也可放在其他任意一台电脑上,但都必须牢记安装 MySQL Server 的机器上的 IP 地址(10.8.16.210)! 至此,数据库和远程访问授权均配置成功。在接下来的章节中,我们将使用 Python 来自动获取相关数据并存入数据库中。

(20) 安装 Python 连接 MySQL 数据库的包 pymysql。回到 Windows 桌面,单击“开始”→Anaconda3(32-bit)→Anaconda Prompt,如图 6.25 所示。

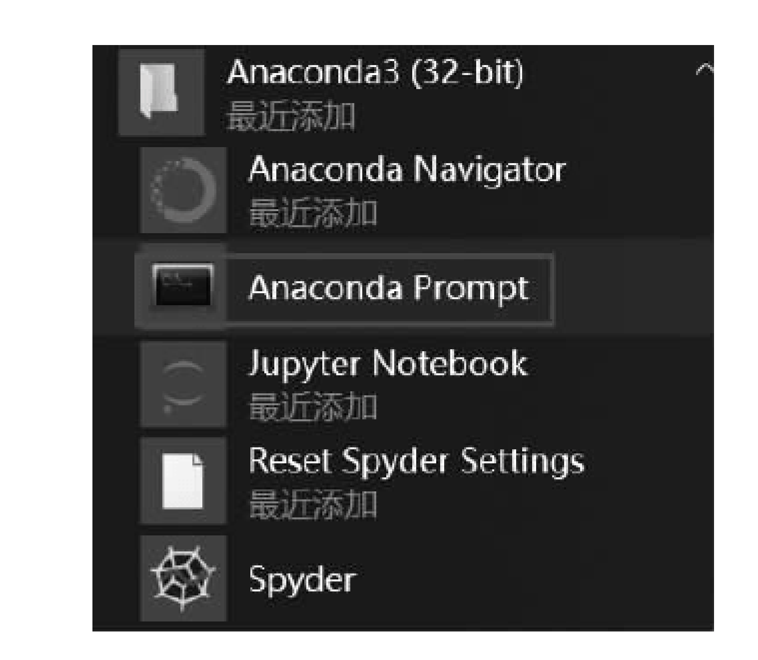


图 6.25　启动 Anaconda Prompt

单击图 6.25 中的“Anaconda Prompt”图标,即可弹出图 6.26 所示的 Windows Shell 窗口。首先输入“conda install pymysql”并按回车键,接着会提示是否继续安装 pymysql,输入“y”并按回车键,即可完成 pymysql 包的安装。

(21) 通过 Wind 代码生成器(WindNavigator)自动生成部分代码。找到 Wind 安装路径,本书以“F:\Wind\Wind.NET.Client\WindNET\bin”为示例,如图 6.27 所示。

双击图 6.27 中的“WindNavigator.exe”可执行文件,弹出图 6.28 所示的代码生成器窗口,并选择编程语言为 Python。

```
(F:\Users\CuiYuZheng\Anaconda3) C:\Users\CuiYuZheng>conda install pymysql
Fetching package metadata .............
Solving package specifications: .

Package plan for installation in environment F:\Users\CuiYuZheng\Anaconda3:

The following NEW packages will be INSTALLED:

    pymysql:  0.8.0-py36_0

The following packages will be UPDATED:

    anaconda: 5.0.1-py36h2419598_2  --> custom-py36h8b40cde_0
    conda:    4.3.30-py36h404fb56_0 --> 4.5.0-py36_0
    pycosat:  0.6.2-py36h06ed0df_1  --> 0.6.3-py36hbfdb8c3_0

Proceed ([y]/n)? y

anaconda-custo 100% |###############################| Time: 0:00:00   1.05 MB/s
pycosat-0.6.3- 100% |###############################| Time: 0:00:00 231.44 kB/s
pymysql-0.8.0- 100% |###############################| Time: 0:00:00 571.85 kB/s
conda-4.5.0-py 100% |###############################| Time: 0:00:00   1.92 MB/s

(F:\Users\CuiYuZheng\Anaconda3) C:\Users\CuiYuZheng>
```

图 6.26　安装 pymysql 包

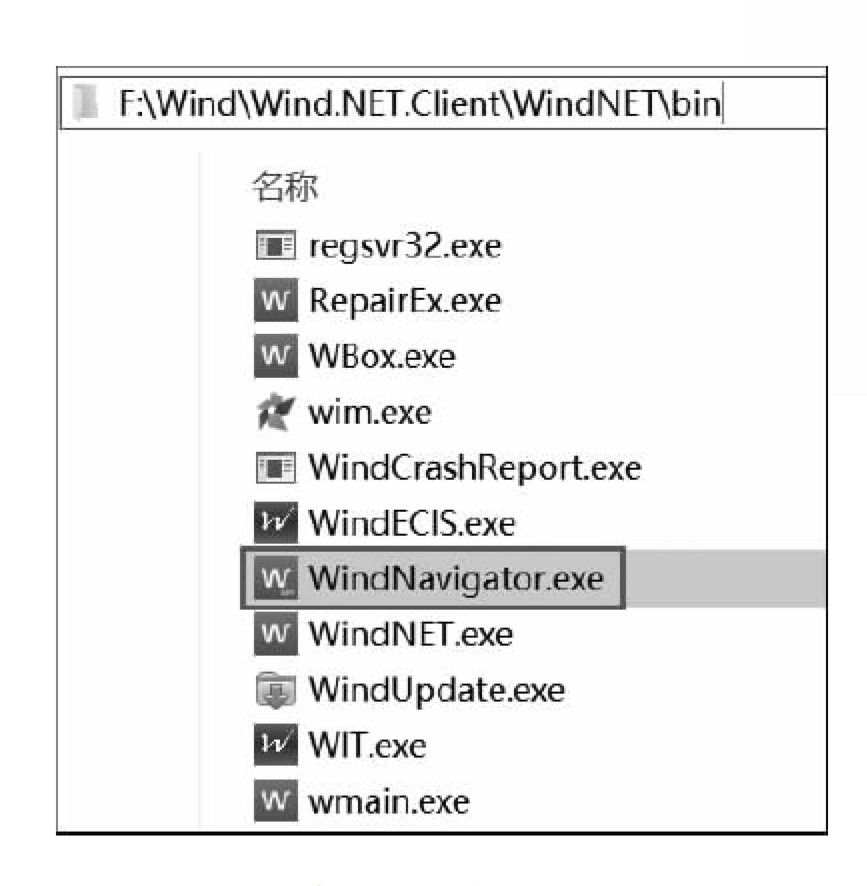


图 6.27　Wind 代码生成器(WindNavigator)

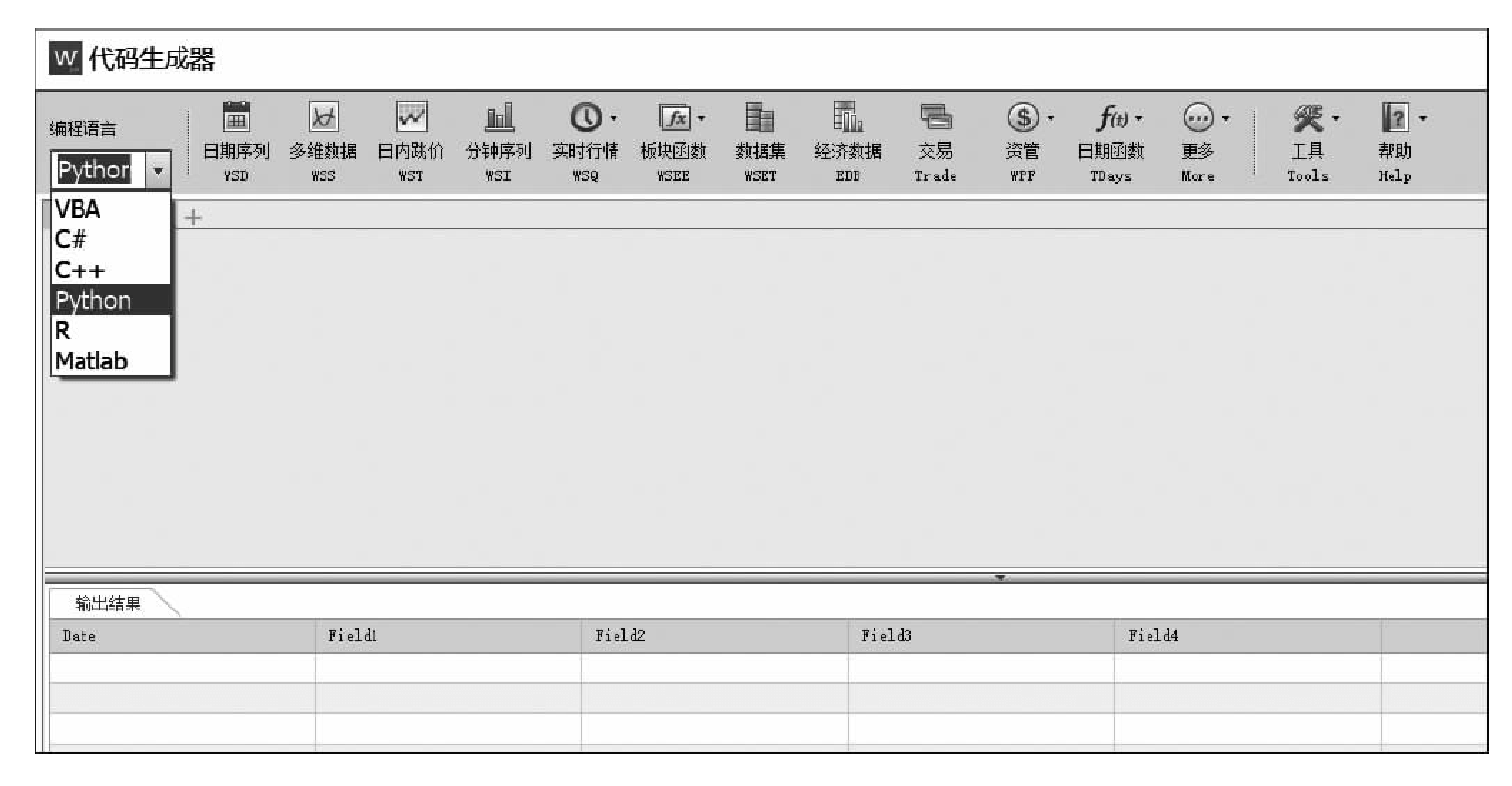
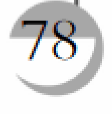


图 6.28　选择编程语言



（22）获取指定板块基础数据的 Python 代码。在打开的代码生成器窗口中，单击“数据集”，会弹出数据集对话框，如图 6.29 所示。



图 6.29　数据集对话框

单击“板块 id”的编辑按钮，弹出“参数设置”对话框。选中需要获取板块数据的名称，此处以获取民营企业（信用债）为例，单击“确定”按钮，如图 6.30 所示。



图 6.30　指定数据集的板块

选择“直接运行”，并单击“确定”按钮，如图 6.31 所示。

单击图 6.31 中的“确定”按钮后，会自动生成获取选定板块数据的 Python

数据集

数据集列表

板块成分
指数权重
指数成分
股票
停牌股票
复牌股票
同企业为其他企业担保
员工持股计划
公司舆情
沪深一级市场
沪深二级市场
沪深公司研究
沪深市场概况
沪深盈利预测
两融交易数据
港股公司研究
债券
可交割券转换因子
最便宜可交割券
回购担保品
债券发行与到期统计
同一个企业所发债券
现金流
期权
期权列表
沪深交易所期权
商品交易所期权
期货
期货合约列表
品种持仓排名
合约持仓排名

<按拼音首字母搜索>

设置参数

参数名称	中文名称	参数值	参数值释义	编...
date	日期	2018-04-08		
sectorid	板块id	1000004555...	民营企业(信...	

显示输出字段

	中文名称	英文名称
☑	日期	date
☑	wind代码	wind_code
☑	证券名称	sec_name

运行设置

○不运行 ◉直接运行

	数据列表	证券代码列表	字段列表	时间列表	错误ID	请求ID
输出变量名	w_wset_data	w_wset_codes	w_wset_fields	w_wset_times	w_wset_errorid	w_wset_reqid

确定 取消

图 6.31　选择“直接运行”

代码，并输出这些代码的运行结果，如图 6.32 所示。

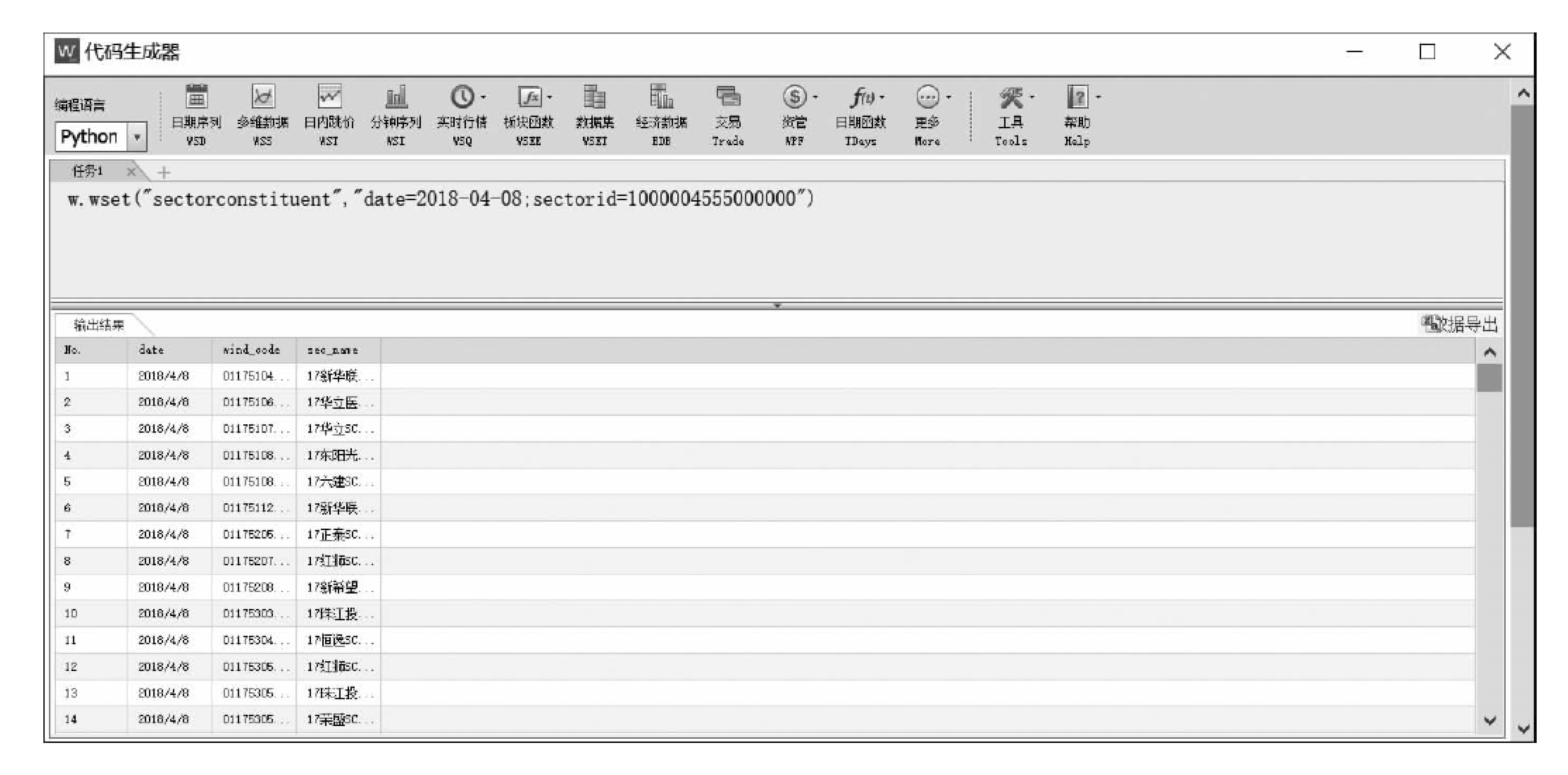


图 6.32　自动生成代码并输出其运行结果

（23）使用 Wind Python 插件（WindPy）获取信用债发行主体的代码和公司名称，以便后续使用这些基础数据抓取其他建模所需数据。首先，需要安装 WindPy 插件。打开 Wind 资讯金融终端，单击“量化”→“修复插件”→“修复 Python 接口”，如图 6.33 所示。

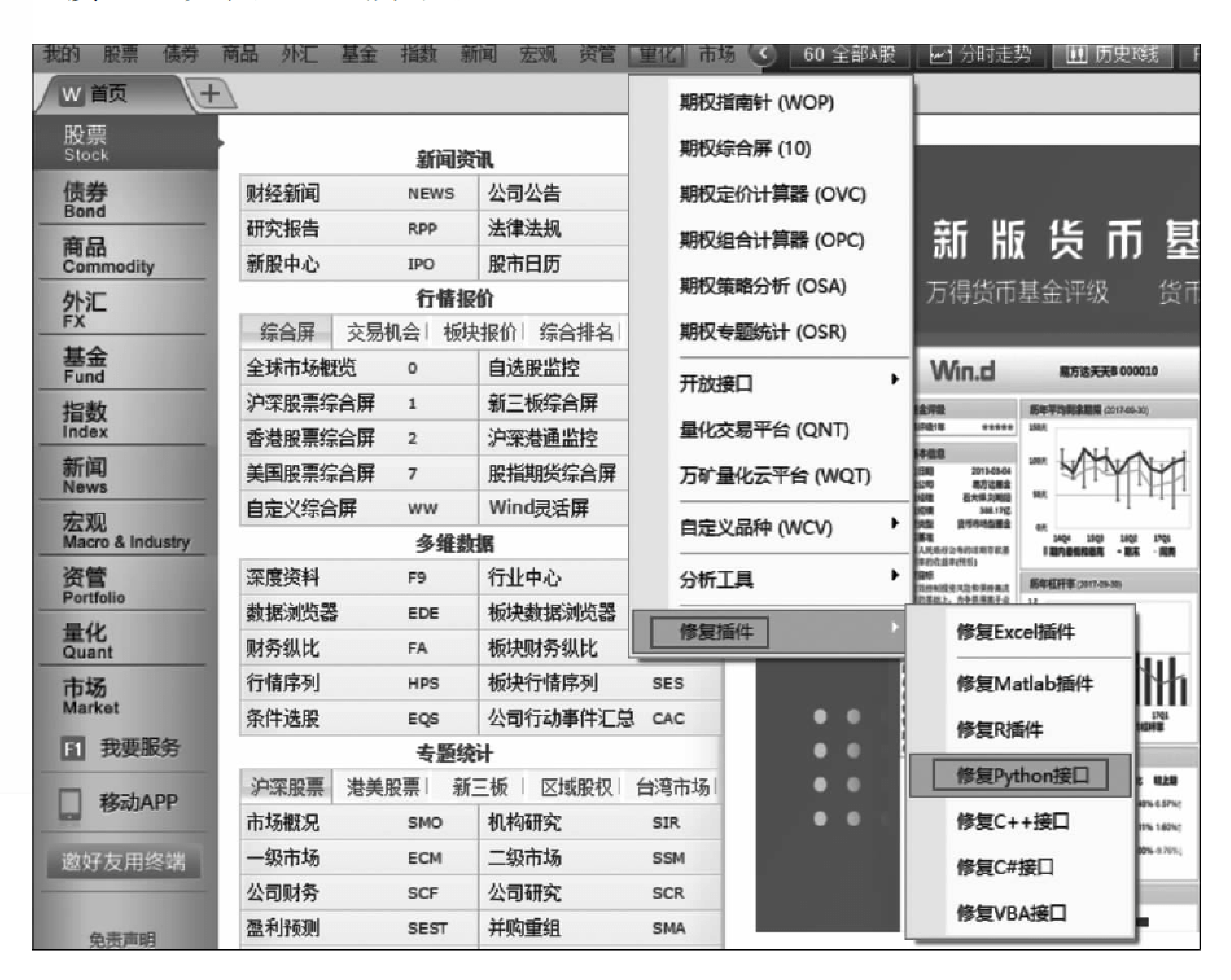


图 6.33　安装 WindPy 插件

WindPy 插件安装完成后，会弹出图 6.34 所示的对话框。



图 6.34　完成安装 WindPy 插件

```
    #因数据量太大,分段读取数据,提升效率
    bond_list=pd.DataFrame()
    slice_size=100 #每段读取 100 只债券代码
    n=int(len(bond_codes)/slice_size)#将所有代码分为 n+1 段

    for i in range(n):
        print(i)#打印出当前获取第几段的数据
        code_list=bond_codes[int(i * slice_size):int((i+1) * slice_
size)]
        codes=",".join(code_list)
        w_wss_data=w.wss(codes,'sec_name,comp_name')#调用 Wind 函数获
取数据
        print(codes)#打印出当前段的债券代码
        if w_wss_data.ErrorCode !=0:
            print(w_wss_data)
            break
        mm=np.mat(w_wss_data.Data)#将获取的数据封装成矩阵
        df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
        df['CODE']=w_wss_data.Codes
        bond_list=bond_list.append(df)#拼接已经打印的债券代码

    print('Final')
    code_list=bond_codes[int(n * slice_size):len(bond_codes)]
    codes=",".join(code_list)
    w_wss_data=w.wss(codes,'sec_name,comp_name')#抓取剩余部分(第 n+1
段)的数据
    print(codes)
    if w_wss_data.ErrorCode !=0:
        print(w_wss_data)
    mm=np.mat(w_wss_data.Data)
    df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
    df['CODE']=w_wss_data.Codes
    bond_list=bond_list.append(df)
    bond_list.index=range(len(bond_list))
    return bond_list #返回所有打印的债券代码

##############Start ###############
w.start() #启动 WindPy 插件
bond_list=get_bond_list() #调用函数抓取债券代码和发行主体公司名称列表
distinct_company_list=pd.DataFrame()
distinct_company_list=distinct_company_list.append(bond_list.loc
[0,])
```

```
#因存在一个主体发行多只债券的情况,为避免获取重复的数据,此处按照公司名称去重
for i in range(len(bond_list)):
    if bond_list.COMP_NAME[i] in list(distinct_company_list.COMP_
NAME):
        continue
    else:
        distinct_company_list=distinct_company_list.append(bond_
list.loc[i,])

del distinct_company_list["SEC_NAME"]
#配置数据库链接
engine= create_engine("mysql+pymysql://test:admin@10.8.16.210:
3306/creditrisk?charset=gbk")
distinct_company_list.to_sql(name='distinct_company_list', con=
engine, if_exists='replace', index=False, index_label=False)#将抓取
的数据,直接保存到 MySQL 数据库
w.stop()#关闭 WindPy 插件
```

本节重点介绍抓取债券代码和公司名称的原理和技术,在接下来的章节中我们将详细介绍使用已经抓取的这些债券代码和公司名称来抓取相关的财务数据,并进行统计检验。

6.2 财务数据对违约状态影响弱显著及 Python 源代码

本节重点介绍使用财务数据检验其对违约状态的影响是否显著的方法,主要包括三部分:抓取违约债券列表并标记为 1;抓取非违约民营企业信用债发行主体的财务数据并标记为 0;计算财务数据对违约状态影响的 P-Value,详细代码如下所示。

```
#载入所需的 Python 包
from WindPy import w
import pandas as pd
import numpy as np
import statsmodels.api as sm
#启动 WindPy 插件
w.start()
#抓取违约债券列表和公司
wsetdata=w.wset("sectorconstituent","date=2018-04-01;sectorid=
1000022486000000")#获取全部违约债券代码
code_list=wsetdata.Data[1]
```

```
codes=code_list[0]

for code in code_list[1:]:
    codes=codes+','+code
#使用获取的违约债券代码,抓取公司名称
w_data=w.wss(codes, "comp_name")

comp_name_data=pd.DataFrame()
comp_name_data['CODE']=w_data.Codes
comp_name_data['COMP_NAME']=w_data.Data[0]
comp_name_data['TARGET']=1 #将违约公司标记为 1
#因存在一个公司发行多只债券的情况,为了避免重复获取数据,先删除重复的公司
unique_comp=comp_name_data.drop_duplicates(['COMP_NAME'])#剔除重名
的公司,只保留第一个
unique_comp.index=range(len(unique_comp))
#抓取所有民营企业债券和公司
bond_data=w.wset("sectorconstituent","date=2018-04-01;sectorid=
1000004555000000")#获取所有民营企业信用债
#bond_code_dup=pd.DataFrame()
#bond_code_dup['CODE']=bond_data.Data[1]
#bond_code_no_dup=bond_code_dup.drop_duplicates()

bond_list=bond_data.Data[1]
bonds=bond_list[0]
#获取非违约民营企业债券的代码
for bond in bond_list[1:]:
    bonds=bonds+','+bond
#使用民营企业债券的代码,抓取公司名称
w_data=w.wss(bonds, "comp_name")

bond_name_data=pd.DataFrame()
bond_name_data['CODE']=w_data.Codes
bond_name_data['COMP_NAME']=w_data.Data[0]
bond_name_data['TARGET']=0 #将正常公司标记为 0

#因存在一个公司发行多只债券的情况,为了避免重复获取数据,先删除重复的公司
unique_bond=bond_name_data.drop_duplicates(['COMP_NAME'])#剔除重名
的公司,只保留第一个
unique_bond.index=range(len(unique_bond))
```

```
#将违约公司和非违约公司合并到一个数据集
for i in range(len(unique_bond)):
    if unique_bond.loc[i,'COMP_NAME'] in list(unique_comp.COMP_NAME):
        print(i)
        unique_bond.loc[i,'TARGET']=1
company_list=unique_bond.append(unique_comp.loc[~unique_comp
['COMP_NAME'].isin(list(unique_bond.COMP_NAME))])
company_list.index=range(len(company_list))

#抓取上述数据集中所有公司的财务数据
company_codes=list(company_list.CODE)
rptDate_list=['20161231']#此处仅以抓取2016年年报的数据为例
index_set=pd.DataFrame()
slice_size=30 #为提升效率,每次抓取30家公司的财务数据
n=int(len(company_codes)/slice_size)
#抓取财务数据
for rptdate in rptDate_list:
    print(rptdate)
    index=0
    for i in range(index,n):
        print(i)
        code_list=company_codes[int(i*slice_size):int((i+1)*slice
_size)]
        codes=",".join(code_list)
        #盈利能力,收益质量,现金流量,资本结构,偿债能力,营运能力,规模效应
         wind_args="rptDate="+str(rptdate)+";tradeDate="+str
(rptdate)+";rptType=1;unit=1;order=1"
        w_wss_data=w.wss(codes,'roe_basic,operateexpensetogr,roa2,
operateincometoebt,deductedprofittoprofit,ocftoassets,currentdebttodebt,
catoassets,ocftodebt,caturn,invturn,net_cash_flows_oper_act,
opprofit,tot_oper_rev', wind_args)
        print(codes)
        if w_wss_data.ErrorCode !=0:
            print(w_wss_data)
            #break;
        mm=np.mat(w_wss_data.Data)
        df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
        df['CODE']=w_wss_data.Codes
        df['rptDate']=[rptdate for j in range(len(code_list))]
        index_set=index_set.append(df)
```

```
    print('Final')
    code_list=company_codes[int(n*slice_size):len(company_codes)]
    codes=",".join(code_list)
    #盈利能力,收益质量,现金流量,资本结构,偿债能力,营运能力,规模效应
    wind_args="rptDate="+str(rptdate)+";tradeDate="+str(rptdate)+";rptType=1;unit=1;order=1"
    w_wss_data=w.wss(codes,'roe_basic,operateexpensetogr,roa2,operateincometoebt,deductedprofittoprofit,ocftoassets,currentdebttodebt,catoassets,ocftodebt,caturn,invturn,net_cash_flows_oper_act,opprofit,tot_oper_rev', wind_args)
    print(codes)
    if w_wss_data.ErrorCode !=0:
        print(w_wss_data)
        break;
    mm=np.mat(w_wss_data.Data)
    df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
    df['CODE']=w_wss_data.Codes
    df['rptDate']=[rptdate for j in range(len(code_list))]
    index_set=index_set.append(df)

    index=0

index_set.index=range(len(index_set))

#去除空缺值过多的指标
name_list=list(index_set)
protect_list=['TOT_OPER_REV']
n=index_set.columns.size
m=index_set.iloc[:,0].size
adjust=0
for i in range(n):
    data_list=index_set.iloc[:,i -adjust]
    na_count=0
    if name_list[i] in protect_list:
        continue
    for j in range(m):
        if data_list[j] !=data_list[j]:
            na_count +=1
    if (na_count/m)>(1/3): #如果该指标的缺失值大于 1/3,删除该指标
```

```
        del index_set[name_list[i]]
        adjust +=1

#去除空缺值过多的公司
n=index_set.columns.size
m=index_set.iloc[:,0].size
adjust=0
for i in range(m):
    data_list=index_set.iloc[i -adjust,:]
    na_count=0
    for j in range(n):
        if data_list[j] !=data_list[j]:
            na_count +=1
    if (na_count/n)>(1/3): #如果该公司的缺失值大于1/3,删除该公司
        index_set=index_set.drop(i)
        adjust +=1

index_set=index_set.loc[~np.isnan(index_set['TOT_OPER_REV'])]
index_set.index=range(len(index_set))

#填充缺失值
complete_set=pd.DataFrame()

for rptdate in rptDate_list:
    print(rptdate)
    data_set=index_set.loc[index_set['rptDate'] ==str(rptdate)]
    boundarys=[data_set['TOT_OPER_REV'].quantile(0),data_set['TOT_
OPER_REV'].quantile(0.25),
                     data_set['TOT_OPER_REV'].quantile(0.5),data_set
['TOT_OPER_REV'].quantile(0.75),
                     data_set['TOT_OPER_REV'].quantile(1)]
    #将所有样本按照营业收入的四分位数,分为大型、中大型、中型、小型公司
    part_1=data_set[data_set['TOT_OPER_REV']<=boundarys[1]]
    part_2=data_set[data_set['TOT_OPER_REV']>boundarys[1]]
    part_2=part_2[part_2['TOT_OPER_REV']<=boundarys[2]]
    part_3=data_set[data_set['TOT_OPER_REV']>boundarys[2]]
    part_3=part_3[part_3['TOT_OPER_REV']<=boundarys[3]]
    part_4=data_set[data_set['TOT_OPER_REV']>boundarys[3]]

    n=part_1.columns.size
```

```
    #如果一个公司属于大型公司,且出现了缺失值,则用其他大型公司该指标的平均数填充该缺失值;其他类型的公司,以此类推。
    part_1_means=part_1.iloc[:,:(n-3)].describe().mean()
    part_2_means=part_2.iloc[:,:(n-3)].describe().mean()
    part_3_means=part_3.iloc[:,:(n-3)].describe().mean()
    part_4_means=part_4.iloc[:,:(n-3)].describe().mean()
    for i in range(n-3):
        #i=4
        data_list=list(part_1.iloc[:,i])
        for j in range(len(part_1)):
            #j=1
            if np.isnan(data_list[j]):
                part_1.iloc[j,i]=part_1_means[i]

        data_list=list(part_2.iloc[:,i])
        for j in range(len(part_2)):
            #j=1
            if np.isnan(data_list[j]):
                part_2.iloc[j,i]=part_2_means[i]

        data_list=list(part_3.iloc[:,i])
        for j in range(len(part_3)):
            #j=1
            if np.isnan(data_list[j]):
                part_3.iloc[j,i]=part_3_means[i]

        data_list=list(part_4.iloc[:,i])
        for j in range(len(part_4)):
            #j=1
            if np.isnan(data_list[j]):
                part_4.iloc[j,i]=part_4_means[i]

    data_set=part_1.append(part_2).append(part_3).append(part_4)
    complete_set=complete_set.append(data_set)

complete_set.index=range(len(complete_set))

#P-Value 计算,统计财务指标对违约状态影响的显著性
col_list=list(complete_set.columns)
X=np.array(complete_set.loc[:,col_list[:-3]])
```

```
marked_set=complete_set.merge(company_list,how='left', on=['CODE'])
marked_set=marked_set.fillna(0)
y=np.array(marked_set.TARGET)
X2=sm.add_constant(X)
est=sm.OLS(y, X2)
est2=est.fit()
print(est2.summary())#输出统计结果检验概述,如图 6.35 所示
```

```
In [559]: print(est2.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.067
Model:                            OLS   Adj. R-squared:                  0.052
Method:                 Least Squares   F-statistic:                     4.644
Date:                Mon, 09 Apr 2018   Prob (F-statistic):           7.23e-07
Time:                        15:01:23   Log-Likelihood:                 327.01
No. Observations:                 726   AIC:                            -630.0
Df Residuals:                     714   BIC:                            -575.0
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          0.0888      0.025      3.553      0.000         0.040     0.138
x1            -0.0017      0.000     -3.712      0.000        -0.003    -0.001
x2            -0.0041      0.001     -5.839      0.000        -0.005    -0.003
x3         -5.697e-05   4.73e-05     -1.203      0.229        -0.000   3.6e-05
x4          5.513e-06      0.001      0.008      0.993        -0.001     0.001
x5             0.0001      0.000      0.311      0.756        -0.001     0.001
x6            -0.0004      0.000     -1.181      0.238        -0.001     0.000
x7             0.0022      0.007      0.330      0.741        -0.011     0.015
x8            -0.0052      0.007     -0.717      0.474        -0.020     0.009
x9          5.539e-07   1.43e-05      0.039      0.969     -2.76e-05  2.87e-05
x10         7.071e-13   2.36e-12      0.300      0.764     -3.92e-12  5.33e-12
x11        -3.486e-12   2.84e-12     -1.229      0.219     -9.06e-12  2.08e-12
==============================================================================
Omnibus:                      798.993   Durbin-Watson:                   0.904
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            32473.690
Skew:                           5.507   Prob(JB):                         0.00
Kurtosis:                      33.858   Cond. No.                     1.54e+10
==============================================================================
```

图 6.35　统计结果检验概述

```
print(list(est2.pvalues))# 输出各指标的 P-Value,如图 6.36 所示
```

```
In [562]: list(est2.pvalues)
Out[562]:
[0.00040516259027454174,
 0.00022171471179346344,
 7.9587125991888631e-09,
 0.22918841456707256,
 0.99337130889183578,
 0.75585478444609344,
 0.23807262329235998,
 0.74127577795974942,
 0.47352032761992957,
 0.96921098632943936,
 0.76410974205413162,
 0.21949317796846091]
```

图 6.36　输出各指标的 P-Value

从图 6.36 的输出结果可见,所有的入模指标中只有前 3 个指标的 P-Value 是小于 0.05 的,也就是说只有前 3 个指标对违约状态的影响是显著的。其他的指标,分别属于指标大类中的现金流量、资本结构、偿债能力、营运能力、规模效应五类,这些类别的指标对违约状况的影响不显著。这也充分说明了,目前国内企业粉饰财报的现象比较普遍,仅仅从财务数据本身无法真实有效地评估企业的信用风险。

6.3 场外数据对违约状态影响强显著及 Python 源代码

由 5.2 节中介绍的应对企业财务数据粉饰的解决方案中的方法,我们通过采用场外的司法、招聘、场外融资、存量债券的到期收益率等真实信息,可有效预测企业的真实信用风险。

由于这些场外真实信息的获取需要较强的 IT 技能,如需要强大的爬虫团队来爬去司法、招聘和场外融资等的数据,然后需要将这些爬取的非结构化数据通过大量的文本挖掘、算法处理等数据预处理技术来提取备选的入模指标,本书就不详述这些场外数据的抓取和预处理方式了。此处,仅以少量作者实战模型开发中的样本作为测试样本,来演示场外数据对违约状态影响的显著程度。样本总体共计 119 个,其中违约样本 20 个、非违约样本 99 个。

```
#载入显著性检验所需的 Python 包
import pandas as pd
import numpy as np
import statsmodels.api as sm
#读取数据
data_set=pd.read_excel("company_data.xlsx")
X=np.array(data_set.iloc[:,1:])
y=np.array(data_set.Target)
#计算 P-Value
X2=sm.add_constant(X)
est=sm.OLS(y, X2)
est2=est.fit()
print(est2.summary())#输出统计结果检验概述,如图 6.37 所示
list(est2.pvalues) # 输出各指标的 P-Value,如图 6.38 所示
```

由图 6.38 所示各指标的 P-Value 检验结果可知,基本所有这些场外指标均小于 0.05,也就是说这些场外数据对违约状态的影响是非常显著的。与图 6.36 财务数据的检验结果相比,场外数据对违约状态的影响明显要显著很多。

OLS Regression Results

Dep. Variable:	y	R-squared:	0.226
Model:	OLS	Adj. R-squared:	0.184
Method:	Least Squares	F-statistic:	5.400
Date:	Wed, 11 Apr 2018	Prob (F-statistic):	6.28e-05
Time:	18:17:33	Log-Likelihood:	-36.644
No. Observations:	118	AIC:	87.29
Df Residuals:	111	BIC:	106.7
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1250	0.034	3.654	0.000	0.057	0.193
x1	0.0442	0.023	1.895	0.061	-0.002	0.090
x2	-0.2978	0.097	-3.086	0.003	-0.489	-0.107
x3	0.0107	0.006	1.792	0.076	-0.001	0.023
x4	-0.2067	0.081	-2.551	0.012	-0.367	-0.046
x5	0.0813	0.032	2.572	0.011	0.019	0.144
x6	0.1387	0.043	3.244	0.002	0.054	0.223
x7	0.0107	0.006	1.792	0.076	-0.001	0.023

Omnibus:	44.462	Durbin-Watson:	1.534
Prob(Omnibus):	0.000	Jarque-Bera (JB):	79.619
Skew:	1.762	Prob(JB):	5.14e-18
Kurtosis:	4.942	Cond. No.	4.89e+15

图 6.37　统计结果检验概述

```
[0.00039584022755041574,
 0.060728360566948525,
 0.002564697861792406,
 0.075851803846473151,
 0.012108065107112084,
 0.011417665644805496,
 0.0015553700647225254,
 0.075851803846474705]
```

图 6.38　输出各指标的 P-Value

6.4　财务粉饰的本福特法则统计识别法及 Python 源代码

本福特法则是由英国物理学家本福特因发现在一些自然数据源中各数字的首位数字出现的频率不满足均匀分布而闻名的。其基本内容是在自然数据源(如信用卡账单、采购记录、财务报表等)生成的数字中，约有 30%的数字的首位数字是 1；首位数字为 2 的数约有 18%；顺序递减，首位数字为 9 的数少于 5%。

精确的数学表述为：在 b 进位制中，以数 n 开头的数字出现的频率满足式(6.1)，数字 1～9 出现的理论频率如表 6.1 所示。

$$F(n)=\log_b(n+1)-\log_b(n) \tag{6.1}$$

表 6.1　数字 1～9 出现的理论频率

数字	频率	数字	频率	数字	频率
1	0.301	4	0.097	7	0.058
2	0.176	5	0.079	8	0.051
3	0.125	6	0.067	9	0.046

其实，本福特法则的理论统计结果，也可用我们日常生活中的例子解释，例如：

(1) 图书馆里大部分书的头几页通常比较脏。因为许多到图书馆看书的人大多只是看书的开头，不喜欢的话就不会再看下去了；把一本书完整看完的人比较少。

(2) 数学书后的对数表、化学书后的一些化学常数、财务课本后的终值、现值系数表等，我们查阅的数据大多在头几页里面。

如果统计的数据足够多，我们会发现，开头是数字 1 的数据最多，大约占了所有数据的 1/3；开头是数字 2 的数据居于其次；剩下的数字的数量依次递减。人口、死亡率、物理和化学常数、棒球统计表、半衰期放射性同位数、物理书中的答案、素数数字以及斐波纳契数列数字中均有这一定律的身影。换句话说，只要是由度量单位制获得的数据都符合本福特法则，因此本福特法则又称为“第一数字定律”。

但有些情况不适用这个定律，这些数字大多是比较随意的，或者是任意指定的，是一些受限数据而不是由度量单位制获得的。例如彩票数字、电话号码、日期和一组人的体重或者身高数据。

本福特法则在企业财报粉饰方面得到了良好的应用，可大概率识别出企业可能存在的财务造假现象。因为如果做假账的人更改了账本上真实的数据，就会使账本上数字出现的频率发生变化，从而偏离“本福特定律”。通常情况下，在那些假账中，数字 5 和 6 是最常见的开头数字，而不是符合定律的数字 1，这就表明伪造者试图在账目中间“隐藏”数据。最为典型的就是美国安然公司“假账”事件。2001 年曾是美国最大的能源交易商、年营业收入近千亿美元、股票市值 700 多亿美元、世界 500 强中排名第七的安然公司突然宣布破产，当时传出了该公司高层管理人员涉嫌做假账的丑闻。一时间，会计造假成了中外关注的焦点。事后人们发现安然公司在 2001 年度到 2002 年度所公布的财务数据不符合“本福特法则”，这些数字的使用频率与这一定律有较大的偏差，这证明了安然公司

的高层领导确实改动过数据。

但是，随着信息技术的不断进步，很多财务造假者对该法则越来越熟悉，粉饰财报的手法也越来越高明，采用该法则识别财务粉饰的难度也逐渐加大。例如，在本节如下的统计计算结果中，已经出现财务危机的“乐视网”的本福特评分却是处于财报优秀行列。已经出现债券违约的发行主体的本福特评分基本处于较低得分，这也说明本福特法则的统计结果还是有一定参考价值的。我们需要强调的是财报粉饰不一定可以直接造成违约，企业违约是由多方面因素共同作用引起的，财报粉饰可能只是其中的一个因素。

本节分为三部分详述本福特法则的统计计算和评分过程，第一部分是信用债发行主体财务相关数据的抓取，第二部分是本福特法则的统计计算，第三部分是使用本福特法则的统计结果对每家公司的财务状况进行评分。

第一部分，信用债发行主体财务相关数据的抓取，Python 代码如下所示。

```
#抓取信用债发行主体的财务数据(资产负债表、现金流量表、利润表),并写入数据库
#载入抓取数据所需的 Python 包
from WindPy import w
import pandas as pd
import numpy as np
from sqlalchemy import create_engine

#抓取资产负债表科目的所有数据
def get_balance(company_codes):
    #因数据量太大,分段读取数据,提升效率
    rptDate_list=['20161231','20151231','20141231','20131231',
'20121231']#抓取最近5年数据
    index_set=pd.DataFrame()
    slice_size=30 #每次抓取30家公司
    n=int(len(company_codes)/slice_size)
    #逐年遍历抓取资产负债表数据
    for rptdate in rptDate_list:
        print(rptdate) #打印抓取哪一年的数据
        index=0
        for i in range(index,n):
            print(i)
            code_list=company_codes[int(i*slice_size):int((i+1)*
slice_size)]
            codes=",".join(code_list)
            #获取资产负债表科目的所有数据
            wind_args="rptDate="+str(rptdate)+";tradeDate="+str
(rptdate)+";rptType=1;unit=1;order=1"
```

```
        w_wss_data=w.wss(codes,'comp_name,monetary_cap,tradable_
fin_assets,notes_rcv,acct_rcv,oth_rcv,prepay,dvd_rcv,int_rcv,
inventories,consumptive_bio_assets,deferred_exp,hfs_assets,non_cur
_assets_due_within_1y,settle_rsrv,loans_to_oth_banks,margin_acct,
prem_rcv,rcv_from_reinsurer,rcv_from_ceded_insur_cont_rsrv,red_
monetary_cap_for_sale,tot_acct_rcv,oth_cur_assets,tot_cur_assets,
fin_assets_avail_for_sale,held_to_mty_invest,invest_real_estate,
long_term_eqy_invest,long_term_rec,fix_assets,proj_matl,const_in_
prog,fix_assets_disp,productive_bio_assets,oil_and_natural_gas_
assets,intang_assets,r_and_d_costs,goodwill,long_term_deferred_
exp,deferred_tax_assets,loans_and_adv_granted,oth_non_cur_assets,
tot_non_cur_assets,cash_deposits_central_bank,agency_bus_assets,
rcv_invest,asset_dep_oth_banks_fin_inst,precious_metals,rcv_ceded_
unearned_prem_rsrv,rcv_ceded_claim_rsrv,rcv_ceded_life_insur_rsrv,
rcv_ceded_lt_health_insur_rsrv,insured_pledge_loan,cap_mrgn_paid,
independent_acct_assets,time_deposits,subr_rec,mrgn_paid,seat_fees
_exchange,clients_cap_deposit,clients_rsrv_settle,oth_assets,
derivative_fin_assets,tot_assets,st_borrow,tradable_fin_liab,notes
_payable,acct_payable,adv_from_cust,empl_ben_payable,taxes_
surcharges_payable,tot_acct_payable,int_payable,dvd_payable,oth_
payable,acc_exp,deferred_inc_cur_liab,hfs_liab,non_cur_liab_due_
within_1y,st_bonds_payable,borrow_central_bank,deposit_received_ib
_deposits,loans_oth_banks,fund_sales_fin_assets_rp,handling_
charges_comm_payable,payable_to_reinsurer,rsrv_insur_cont,acting_
trading_sec,acting_uw_sec,oth_cur_liab,tot_cur_liab,lt_borrow,
bonds_payable,lt_payable,lt_empl_ben_payable,specific_item_
payable,provisions,deferred_tax_liab,deferred_inc_non_cur_liab,oth
_non_cur_liab,tot_non_cur_liab,liab_dep_oth_banks_fin_inst,agency_
bus_liab,cust_bank_dep,claims_payable,dvd_payable_insured,deposit_
received,insured_deposit_invest,unearned_prem_rsrv,out_loss_rsrv,
life_insur_rsrv,lt_health_insur_v,independent_acct_liab,prem_
received_adv,pledge_loan,st_finl_inst_payable,oth_liab,derivative_
fin_liab,tot_liab,cap_stk,other_equity_instruments,other_equity_
instruments_PRE,cap_rsrv,surplus_rsrv,undistributed_profit,tsy_
stk,other_compreh_inc_bs,special_rsrv,prov_nom_risks,cnvd_diff_
foreign_curr_stat,unconfirmed_invest_loss_bs,minority_int,eqy_
belongto_parcomsh,tot_equity,tot_liab_shrhldr_eqy',wind_args)#抓取
资产负债表中所有科目的数据
        print(codes)#打印抓取公司的债券代码
        if w_wss_data.ErrorCode !=0:
            print(w_wss_data)
            break;
```

```
            mm=np.mat(w_wss_data.Data)
            df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
            df['CODE']=w_wss_data.Codes
            df['rptDate']=[rptdate for j in range(len(code_list))]
            df.to_sql(name='balance', con=engine, if_exists='append',
index=False, index_label=False)#将抓取的数据存储到数据库中,数据库表名
称为 balance

        print('Final')#抓取最后一段的资产负债表数据
        code_list=company_codes[int(n * slice_size):len(company_
codes)]
        codes=",".join(code_list)
        #获取资产负债表科目的所有数据
        wind_args="rptDate="+str(rptdate)+";tradeDate="+str
(rptdate)+";rptType=1;unit=1;order=1"
        w_wss_data=w.wss(codes,'comp_name,monetary_cap,tradable_
fin_assets,notes_rcv,acct_rcv,oth_rcv,prepay,dvd_rcv,int_rcv,
inventories,consumptive_bio_assets,deferred_exp,hfs_assets,non_cur
_assets_due_within_1y,settle_rsrv,loans_to_oth_banks,margin_acct,
prem_rcv,rcv_from_reinsurer,rcv_from_ceded_insur_cont_rsrv,red_
monetary_cap_for_sale,tot_acct_rcv,oth_cur_assets,tot_cur_assets,
fin_assets_avail_for_sale,held_to_mty_invest,invest_real_estate,
long_term_eqy_invest,long_term_rec,fix_assets,proj_matl,const_in_
prog,fix_assets_disp,productive_bio_assets,oil_and_natural_gas_
assets,intang_assets,r_and_d_costs,goodwill,long_term_deferred_
exp,deferred_tax_assets,loans_and_adv_granted,oth_non_cur_assets,
tot_non_cur_assets,cash_deposits_central_bank,agency_bus_assets,
rcv_invest,asset_dep_oth_banks_fin_inst,precious_metals,rcv_ceded_
unearned_prem_rsrv,rcv_ceded_claim_rsrv,rcv_ceded_life_insur_rsrv,
rcv_ceded_lt_health_insur_rsrv,insured_pledge_loan,cap_mrgn_paid,
independent_acct_assets,time_deposits,subr_rec,mrgn_paid,seat_fees
_exchange,clients_cap_deposit,clients_rsrv_settle,oth_assets,
derivative_fin_assets,tot_assets,st_borrow,tradable_fin_liab,notes
_payable,acct_payable,adv_from_cust,empl_ben_payable,taxes_
surcharges_payable,tot_acct_payable,int_payable,dvd_payable,oth_
payable,acc_exp,deferred_inc_cur_liab,hfs_liab,non_cur_liab_due_
within_1y,st_bonds_payable,borrow_central_bank,deposit_received_ib
_deposits,loans_oth_banks,fund_sales_fin_assets_rp,handling_
charges_comm_payable,payable_to_reinsurer,rsrv_insur_cont,acting_
trading_sec,acting_uw_sec,oth_cur_liab,tot_cur_liab,lt_borrow,
bonds_payable,lt_payable,lt_empl_ben_payable,specific_item_
payable,provisions,deferred_tax_liab,deferred_inc_non_cur_liab,oth
```

```
_non_cur_liab,tot_non_cur_liab,liab_dep_oth_banks_fin_inst,agency_
bus_liab,cust_bank_dep,claims_payable,dvd_payable_insured,deposit_
received,insured_deposit_invest,unearned_prem_rsrv,out_loss_rsrv,
life_insur_rsrv,lt_health_insur_v,independent_acct_liab,prem_
received_adv,pledge_loan,st_finl_inst_payable,oth_liab,derivative_
fin_liab,tot_liab,cap_stk,other_equity_instruments,other_equity_
instruments_PRE,cap_rsrv,surplus_rsrv,undistributed_profit,tsy_
stk,other_compreh_inc_bs,special_rsrv,prov_nom_risks,cnvd_diff_
foreign_curr_stat,unconfirmed_invest_loss_bs,minority_int,eqy_
belongto_parcomsh,tot_equity,tot_liab_shrhldr_eqy',wind_args) #抓取
资产负债表中所有科目的数据
        print(codes)
        if w_wss_data.ErrorCode !=0:
            print(w_wss_data)
            break;
        mm=np.mat(w_wss_data.Data)
        df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
        df['CODE']=w_wss_data.Codes
        df['rptDate']=[rptdate for j in range(len(code_list))]
        df.to_sql(name='balance', con=engine, if_exists='append',
index=False, index_label=False)
        index=0

#抓取现金流量表科目的所有数据
def get_cashflow(company_codes):
    #因数据量太大,分段读取数据,提升效率
    rptDate_list=['20161231','20151231','20141231','20131231','
20121231']
    index_set=pd.DataFrame()
    slice_size=30
    n=int(len(company_codes)/slice_size)
    #逐年遍历抓取现金流量表数据
    for rptdate in rptDate_list:
        print(rptdate) #打印抓取哪一年的数据
        index=0
        for i in range(index,n):
            print(i)
            code_list=company_codes[int(i*slice_size):int((i+1)*
slice_size)]
            codes=",".join(code_list)
            #获取现金流量表科目的所有数据
            wind_args="rptDate="+str(rptdate)+";tradeDate="+str
```

```
(rptdate)+";rptType=1;unit=1;order=1"
         w_wss_data=w.wss(codes,'comp_name,cash_recp_sg_and_rs,
recp_tax_rends,other_cash_recp_ral_oper_act,net_incr_insured_dep,
net_incr_dep_cob,net_incr_loans_central_bank,net_incr_fund_borr_
ofi,net_incr_int_handling_chrg,cash_recp_prem_orig_inco,net_cash_
received_reinsu_bus,net_incr_disp_tfa,net_incr_disp_fin_assets_
avail,net_incr_loans_other_bank,net_incr_repurch_bus_fund,net_cash
_from_seurities,stot_cash_inflows_oper_act,net_incr_lending_fund,
net_fina_instruments_measured_at_fmv,cash_pay_goods_purch_serv_
rec,cash_pay_beh_empl,pay_all_typ_tax,other_cash_pay_ral_oper_act,
net_incr_clients_loan_adv,net_incr_dep_cbob,cash_pay_claims_orig_
inco,handling_chrg_paid,comm_insur_plcy_paid,stot_cash_outflows_
oper_act,net_cash_flows_oper_act,cash_recp_disp_withdrwl_invest,
cash_recp_return_invest,net_cash_recp_disp_fiolta,net_cash_recp_
disp_sobu,other_cash_recp_ral_inv_act,stot_cash_inflows_inv_act,
cash_pay_acq_const_fiolta,cash_paid_invest,net_incr_pledge_loan,
net_cash_pay_aquis_sobu,other_cash_pay_ral_inv_act,stot_cash_
outflows_inv_act,net_cash_flows_inv_act,cash_recp_cap_contrib,cash
_rec_saims,cash_recp_borrow,other_cash_recp_ral_fnc_act,proc_issue
_bonds,stot_cash_inflows_fnc_act,cash_prepay_amt_borr,cash_pay_
dist_dpcp_int_exp,dvd_profit_paid_sc_ms,other_cash_pay_ral_fnc_
act,stot_cash_outflows_fnc_act,net_cash_flows_fnc_act,eff_fx_flu_
cash,net_incr_cash_cash_equ_dm,cash_cash_equ_beg_period,cash_cash_
equ_end_period,net_profit_cs,prov_depr_assets,depr_fa_coga_dpba,
amort_intang_assets,amort_lt_deferred_exp,decr_deferred_exp,incr_
acc_exp,loss_disp_fiolta,loss_scr_fa,loss_fv_chg,fin_exp_cs,invest
_loss,decr_deferred_inc_tax_assets,incr_deferred_inc_tax_liab,decr
_inventories,decr_oper_payable,incr_oper_payable,unconfirmed_
invest_loss_cs,others,im_net_cash_flows_oper_act,conv_debt_into_
cap,conv_corp_bonds_due_within_1y,fa_fnc_leases,end_bal_cash,beg_
bal_cash,end_bal_cash_equ,beg_bal_cash_equ,net_incr_cash_cash_equ_
im',wind_args)
         print(codes) #打印抓取公司的债券代码
         if w_wss_data.ErrorCode !=0:
            print(w_wss_data)
            break;
         mm=np.mat(w_wss_data.Data)
         df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
         df['CODE']=w_wss_data.Codes
         df['rptDate']=[rptdate for j in range(len(code_list))]
         df.to_sql(name='cashflow', con=engine, if_exists='
append', index=False, index_label=False) #将抓取的数据存储到数据库中，
```

数据库表名称为 cashflow

```
        print('Final') #抓取最后一段的资产负债表数据
        code_list=company_codes[int(n * slice_size):len(company_codes)]
        codes=",".join(code_list)
        #获取现金流量表科目的所有数据
        wind_args="rptDate="+str(rptdate)+";tradeDate="+str(rptdate)+";rptType=1;unit=1;order=1"
        w_wss_data=w.wss(codes,'comp_name,cash_recp_sg_and_rs,recp_tax_rends,other_cash_recp_ral_oper_act,net_incr_insured_dep,net_incr_dep_cob,net_incr_loans_central_bank,net_incr_fund_borr_ofi,net_incr_int_handling_chrg,cash_recp_prem_orig_inco,net_cash_received_reinsu_bus,net_incr_disp_tfa,net_incr_disp_fin_assets_avail,net_incr_loans_other_bank,net_incr_repurch_bus_fund,net_cash_from_seurities,stot_cash_inflows_oper_act,net_incr_lending_fund,net_fina_instruments_measured_at_fmv,cash_pay_goods_purch_serv_rec,cash_pay_beh_empl,pay_all_typ_tax,other_cash_pay_ral_oper_act,net_incr_clients_loan_adv,net_incr_dep_cbob,cash_pay_claims_orig_inco,handling_chrg_paid,comm_insur_plcy_paid,stot_cash_outflows_oper_act,net_cash_flows_oper_act,cash_recp_disp_withdrwl_invest,cash_recp_return_invest,net_cash_recp_disp_fiolta,net_cash_recp_disp_sobu,other_cash_recp_ral_inv_act,stot_cash_inflows_inv_act,cash_pay_acq_const_fiolta,cash_paid_invest,net_incr_pledge_loan,net_cash_pay_aquis_sobu,other_cash_pay_ral_inv_act,stot_cash_outflows_inv_act,net_cash_flows_inv_act,cash_recp_cap_contrib,cash_rec_saims,cash_recp_borrow,other_cash_recp_ral_fnc_act,proc_issue_bonds,stot_cash_inflows_fnc_act,cash_prepay_amt_borr,cash_pay_dist_dpcp_int_exp,dvd_profit_paid_sc_ms,other_cash_pay_ral_fnc_act,stot_cash_outflows_fnc_act,net_cash_flows_fnc_act,eff_fx_flu_cash,net_incr_cash_cash_equ_dm,cash_cash_equ_beg_period,cash_cash_equ_end_period,net_profit_cs,prov_depr_assets,depr_fa_coga_dpba,amort_intang_assets,amort_lt_deferred_exp,decr_deferred_exp,incr_acc_exp,loss_disp_fiolta,loss_scr_fa,loss_fv_chg,fin_exp_cs,invest_loss,decr_deferred_inc_tax_assets,incr_deferred_inc_tax_liab,decr_inventories,decr_oper_payable,incr_oper_payable,unconfirmed_invest_loss_cs,others,im_net_cash_flows_oper_act,conv_debt_into_cap,conv_corp_bonds_due_within_1y,fa_fnc_leases,end_bal_cash,beg_bal_cash,end_bal_cash_equ,beg_bal_cash_equ,net_incr_cash_cash_equ_im',wind_args)
        print(codes)
        if w_wss_data.ErrorCode !=0:
```

```
            print(w_wss_data)
            break;
        mm=np.mat(w_wss_data.Data)
        df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
        df['CODE']=w_wss_data.Codes
        df['rptDate']=[rptdate for j in range(len(code_list))]
        df.to_sql(name='cashflow', con=engine, if_exists='append',
index=False, index_label=False)
        #index_set=index_set.append(df)
        #index_set.index=range(len(index_set))
        index=0

#抓取利润表科目的所有数据
def get_income(company_codes):
    #因数据量太大,分段读取数据,提升效率
    rptDate_list=['20161231','20151231','20141231','20131231',
'20121231']
    index_set=pd.DataFrame()
    slice_size=30
    n=int(len(company_codes)/slice_size)
    #逐年遍历抓取利润表数据
    for rptdate in rptDate_list:
        print(rptdate) #打印抓取哪一年的数据
        index=0
        for i in range(index,n):
            print(i)
            code_list=company_codes[int(i*slice_size):int((i+1)*
slice_size)]
            codes=",".join(code_list)
            #获取利润表科目的所有数据
            wind_args="rptDate="+str(rptdate)+";tradeDate="+str
(rptdate)+";rptType=1;unit=1;order=1"
            w_wss_data=w.wss(codes,'comp_name,tot_oper_rev,oper_
rev,int_inc,insur_prem_unearned,handling_chrg_comm_inc,tot_prem_
inc,reinsur_inc,prem_ceded,unearned_prem_rsrv_withdraw,net_inc_
agencybusiness, net _ inc _ underwriting - business, net _ inc _
customerasset-managementbusiness,other_oper_inc,net_int_inc,net_
fee_and_commission_inc,net_other_oper_inc,tot_oper_cost,oper_cost,
int_exp,handling_chrg_comm_exp,oper_exp,taxes_surcharges_ops,
selling_dist_exp,gerl_admin_exp,fin_exp_is,impair_loss_assets,
prepay_surr,net_claim_exp,net_insur_cont_rsrv,dvd_exp_insured,
reinsurance_exp,claim_exp_recoverable,Insur_rsrv_recoverable,
```

```
reinsur_exp_recoverable,other_oper_exp,net_inc_other_ops,net_gain_
chg_fv,net_invest_inc,inc_invest_assoc_jv_entp,net_gain_fx_trans,
opprofit,non_oper_rev,non_oper_exp,net_loss_disp_noncur_asset,tot_
profit,tax,unconfirmed_invest_loss_is,net_profit_is,minority_int_
inc,np_belongto_parcomsh,eps_basic_is,eps_diluted_is,other_compreh
_inc,tot_compreh_inc,tot_compreh_inc_min_shrhldr,tot_compreh_inc_
parent_comp',wind_args)
            print(codes) #打印抓取公司的债券代码
            if w_wss_data.ErrorCode !=0:
                print(w_wss_data)
                break;
            mm=np.mat(w_wss_data.Data)
            df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
            df['CODE']=w_wss_data.Codes
            df['rptDate']=[rptdate for j in range(len(code_list))]
            df.to_sql(name='income', con=engine, if_exists='append',
index=False, index_label=False) #将抓取的数据存储到数据库中,数据库表名
称为 income
            #index_set=index_set.append(df)

        print('Final') #抓取最后一段的利润表数据
        code_list=company_codes[int(n*slice_size):len(company_
codes)]
        codes=",".join(code_list)
        #获取利润表科目的所有数据
        wind_args="rptDate="+str(rptdate)+";tradeDate="+str
(rptdate)+";rptType=1;unit=1;order=1"
        w_wss_data=w.wss(codes,'comp_name,tot_oper_rev,oper_rev,int
_inc,insur_prem_unearned,handling_chrg_comm_inc,tot_prem_inc,
reinsur_inc,prem_ceded,unearned_prem_rsrv_withdraw,net_inc_
agencybusiness,net_inc_underwriting-business,net_inc_
customerasset-managementbusiness,other_oper_inc,net_int_inc,net_
fee_and_commission_inc,net_other_oper_inc,tot_oper_cost,oper_cost,
int_exp,handling_chrg_comm_exp,oper_exp,taxes_surcharges_ops,
selling_dist_exp,gerl_admin_exp,fin_exp_is,impair_loss_assets,
prepay_surr,net_claim_exp,net_insur_cont_rsrv,dvd_exp_insured,
reinsurance_exp,claim_exp_recoverable,Insur_rsrv_recoverable,
reinsur_exp_recoverable,other_oper_exp,net_inc_other_ops,net_gain_
chg_fv,net_invest_inc,inc_invest_assoc_jv_entp,net_gain_fx_trans,
opprofit,non_oper_rev,non_oper_exp,net_loss_disp_noncur_asset,tot_
profit,tax,unconfirmed_invest_loss_is,net_profit_is,minority_int_
inc,np_belongto_parcomsh,eps_basic_is,eps_diluted_is,other_compreh
```

```
_inc,tot_compreh_inc,tot_compreh_inc_min_shrhldr,tot_compreh_inc_
parent_comp',wind_args)
        print(codes)
        if w_wss_data.ErrorCode !=0:
            print(w_wss_data)
            break;
        mm=np.mat(w_wss_data.Data)
        df=pd.DataFrame(mm.T, columns=w_wss_data.Fields)
        df['CODE']=w_wss_data.Codes
        df['rptDate']=[rptdate for j in range(len(code_list))]
        df.to_sql(name='income', con=engine, if_exists='append',
index=False, index_label=False)
        #index_set=index_set.append(df)
        #index_set.index=range(len(index_set))
        index=0

#抓取所有财务数据(资产负债表、现金流量表、利润表)并存到数据库中
w.start()
engine=create_engine("mysql+pymysql://test:admin@ 10.8.16.210:
3306/creditrisk?charset=gbk")
distinct_company_list=pd.read_sql_table('distinct_company_list',
con=engine)
company_codes=list(distinct_company_list.CODE)#获取在第一节中抓取的
债券代码
get_balance(company_codes)#抓取这些债券的资产负债表会计科目中的所有数
据,并存到数据库中,数据库表名为 balance
get_cashflow(company_codes)#抓取这些债券的现金流量表会计科目中的所有数
据,并存到数据库中,数据库表名为 cashflow
get_income(company_codes)#抓取这些债券的利润表会计科目中的所有数据,并
存到数据库中,数据库表名为 income
w.stop()#关闭 WindPy 插件
```

第二部分,本福特法则的统计计算,Python 代码如下所示。

```
#加载本福特法则统计计算所需的 Python 包
import pandas as pd
import numpy as np
import time
import pymysql
#设置时间的起点,以便于计算程序运行的时间
time_start=time.time()
#数据库配置,以抓取存储在数据库中的财务数据
config={
```

```
        'host':' 10.8.16.210',
        'port':3306,
        'user':'test',
        'passwd':'admin',
        'db':'creditrisk',
        'charset':'gbk',
        'cursorclass':pymysql.cursors.DictCursor
        }

#本福特法则计算函数
def benfordslaw_calculation(merge_set):
    merge_set=merge_set.fillna(0)
    CODE=merge_set.CODE
    COMP_NAME=merge_set.COMP_NAME
    merge_set=merge_set.drop(['CODE','COMP_NAME'], axis=1)
    merge_set=merge_set.astype(str)
    #初始化变量的统计结果
    num_counts=[[0 for i in range(9)] for i in range(len(merge_set.ix
[:,1]))]
    total_counts=[0 for i in range(len(merge_set.ix[:,1]))]

digit=0
#使用循环统计 1-9 中数字出现的次数
    for i in range(len(merge_set.ix[:,1])):
        print("counting object: "+str(i))
        for j in range(len(merge_set.ix[i,:])):
            if merge_set.ix[i,j][0] !='0':
                total_counts[i]=total_counts[i]+1
                if merge_set.ix[i,j][digit] =='1':
                    num_counts[i][0]=num_counts[i][0]+1
                elif merge_set.ix[i,j][digit] =='2':
                    num_counts[i][1]=num_counts[i][1]+1
                elif merge_set.ix[i,j][digit] =='3':
                    num_counts[i][2]=num_counts[i][2]+1
                elif merge_set.ix[i,j][digit] =='4':
                    num_counts[i][3]=num_counts[i][3]+1
                elif merge_set.ix[i,j][digit] =='5':
                    num_counts[i][4]=num_counts[i][4]+1
                elif merge_set.ix[i,j][digit] =='6':
                    num_counts[i][5]=num_counts[i][5]+1
                elif merge_set.ix[i,j][digit] =='7':
                    num_counts[i][6]=num_counts[i][6]+1
```

```
                elif merge_set.ix[i,j][digit] =='8':
                    num_counts[i][7]=num_counts[i][7]+1
                elif merge_set.ix[i,j][digit] =='9':
                    num_counts[i][8]=num_counts[i][8]+1
            else:
                continue

    #计算每个数字出现的频率
    for i in range(len(num_counts)):
        if sum(num_counts[i]) !=0:
            p_counts=devide_list(num_counts[i], sum(num_counts[i]))
            for j in range(9):
                num_counts[i][j]=p_counts[j]

    counts_diff=[[0 for i in range(9)] for i in range(len(merge_set.ix
[:,1]))]
    #本福特法则理论值
    benfords_dis=[30.1, 17.6, 12.5, 9.7, 7.9, 6.7, 5.8, 5.1, 4.6]
    #计算实际统计结果跟理论值的差异
    for i in range(len(num_counts)):
        for j in range(9):
            counts_diff[i][j]=num_counts[i][j] -benfords_dis[j]/100

    diff_quadratic_sum=[0 for i in range(len(merge_set.ix[:,1]))]
    #计算每个数字的统计频率跟理论值的误差平方和
    for i in range(len(counts_diff)):
        for j in range(9):
            diff_quadratic_sum[i]=diff_quadratic_sum[i]+(counts_
diff[i][j] * counts_diff[i][j])

    #封装本福特法则计算结果
    benford=pd.DataFrame(num_counts, index=CODE, columns=['1','2',
'3','4','5','6','7','8','9'])
    benford['diff_quadratic_sum']=diff_quadratic_sum
    benford['total_counts']=total_counts
    benford['COMP_NAME']=list(COMP_NAME)
    benford=benford.sort_values(by=['diff_quadratic_sum'])
    return benford

#计算频率的函数
def devide_list(input_list,y):
    def f(x):
```

```
            return x / y
        return list(map(f,input_list))

#主函数入口
if __name__=="__main__":
    try:
        con=pymysql.connect(**config)#连接数据库
        with con.cursor() as cursor:
            query="SELECT * FROM credit_management.distinct_company_
list;"
            cursor.execute(query)
            result=cursor.fetchall()#抓取需要下载财务数据的公司列表
    finally:
        con.close

    distinct_company_list=pd.DataFrame(result)

    balance=pd.DataFrame(np.copy(distinct_company_list),columns=
['CODE','COMP_NAME'])
    cashflow=pd.DataFrame(np.copy(distinct_company_list),columns=
['CODE','COMP_NAME'])
    income=pd.DataFrame(np.copy(distinct_company_list),columns=
['CODE','COMP_NAME'])

    for year in range(2012,2017):
        date=str(year)+"1231"
        print(date)
        try:
            con=pymysql.connect(**config)
            with con.cursor() as cursor:
                query="SELECT * FROM credit_management.balance WHERE
rptdate='"+date+"';"
                cursor.execute(query)
                result=cursor.fetchall()#从数据库中抓取资产负债表数据
                balance_x=pd.DataFrame(result)
                del balance_x["rptdate"]
                balance=pd.merge(balance, balance_x, how='left', on=
['CODE','COMP_NAME'])

                query="SELECT * FROM credit_management.cashflow
WHERE rptdate='"+date+"';"
                cursor.execute(query)
```

```
                result=cursor.fetchall()#从数据库中抓取现金流量表数据
                cashflow_x=pd.DataFrame(result)
                del cashflow_x["rptdate"]
                cashflow=pd.merge(cashflow, cashflow_x, how='left',
on=['CODE','COMP_NAME'])

                query="SELECT * FROM credit_management.income WHERE
rptdate='"+date+"';"
                cursor.execute(query)
                result=cursor.fetchall()#从数据库中抓取利润表数据
                income_x=pd.DataFrame(result)
                del income_x["rptdate"]
                income=pd.merge(income, income_x, how='left', on=
['CODE','COMP_NAME'])
        finally:
            con.close

    #调用函数计算财务数据的本福特法则统计结果
    #合并统计资产负债表、现金流量表、利润表的财务数据
    merge_set=pd.merge(balance, cashflow, how='left', on=['CODE',
'COMP_NAME'])
    merge_set=pd.merge(merge_set, income, how='left', on=['CODE',
'COMP_NAME'])
    #计算合并数据的本福特法则统计数字
    benford_total=benfordslaw_calculation(merge_set)
    output_dir="D:/TMP/benford_total.csv"
    benford_total.to_csv(output_dir, index=True, sep=',')#输出到本地
    #计算资产负债表的本福特法则统计数字
    benford_balance=benfordslaw_calculation(balance)
    output_dir="D:/TMP/benford_balance.csv"
    benford_balance.to_csv(output_dir, index=True, sep=',') #输出到本地
    #计算现金流量表的本福特法则统计数字
    benford_cashflow=benfordslaw_calculation(cashflow)
    output_dir="D:/TMP/benford_cashflow.csv"
    benford_cashflow.to_csv(output_dir, index=True, sep=',') #输出到本地
    #计算利润表的本福特法则统计数字
    benford_income=benfordslaw_calculation(income)
    output_dir="D:/TMP/benford_income.csv"
    benford_income.to_csv(output_dir, index=True, sep=',') #输出到本地
    #计算以上所有计算花费的时间
    time_end=time.time()
    time_used=(time_end-time_start)/60
```

```
    print("time_used: "+str(time_used))#输出耗时
```

第三部分，使用本福特法则的统计结果对每家公司的财务状况进行评分，评分的基本原理和步骤主要包括：①计算每家公司所有财务数据（三张报表合并的所有数据）的首位数字统计频率，并计算其跟理论概率的误差平方和；②分别计算每家公司三张财报的首位数字统计频率，并计算其跟理论概率的误差平方和；③将上述四列误差平方和分别计算分位数，并分成 11 段；④将分位数得到的 11 段，从小到大分别附上 1.0，1.1，1.2，…，2.0 的权重；⑤将每列的得分转换为 0～10 分，且 10 分为最高分；⑥计算这四列得分的均值，即为该公司的本福特评分结果，评分越高说明财务质量越好，反之则越差。

本福特评分的 Python 代码如下所示。

```
#使用本福特法则的统计结果，对各公司的财务状况进行评分
import pandas as pd
import numpy as np
import csv

#读取 csv 文件
def read_csv(input_dir):
    csvFile=open(input_dir, "r")
    reader=csv.reader(csvFile)  #返回的是迭代类型
    data=[]
    for item in reader:
        #print(item)
        data.append(item)

    csvFile.close()
    return data

#计算分位数
def quantile(data_list,seq_num=4):
    quantile_list=[]
    for i in range(seq_num+1):
        percentage=(i/seq_num)*100
        quantile_list.append(np.percentile(data_list,percentage))
    return quantile_list

#读取三张报表合并数字的本福特统计结果跟理论结果的误差平方和
input_dir="D:/TMP/benford_total.csv"
benford_total=read_csv(input_dir)
benford_total=pd.DataFrame(benford_total[1:], columns=benford_
```

```
total[0])
benford_total=benford_total.loc[:,["COMP_NAME","total_counts","diff_quadratic_sum"]]
benford_total_boundarys=quantile(benford_total["diff_quadratic_sum"].astype(float),11)
benford_total_boundarys=benford_total_boundarys[1:11]
#读取资产负债表数字的本福特统计结果跟理论结果的误差平方和
input_dir="D:/TMP/benford_balance.csv"
benford_balance=read_csv(input_dir)
benford_balance=pd.DataFrame(benford_balance[1:],columns=benford_balance[0])
benford_balance=benford_balance.loc[:,["COMP_NAME","diff_quadratic_sum"]]
benford_balance_boundarys=quantile(benford_balance["diff_quadratic_sum"].astype(float),11)
benford_balance_boundarys=benford_balance_boundarys[1:11]
#读取现金流量表数字的本福特统计结果跟理论结果的误差平方和
input_dir="D:/TMP/benford_cashflow.csv"
benford_cashflow=read_csv(input_dir)
benford_cashflow=pd.DataFrame(benford_cashflow[1:],columns=benford_cashflow[0])
benford_cashflow=benford_cashflow.loc[:,["COMP_NAME","diff_quadratic_sum"]]
benford_cashflow_boundarys=quantile(benford_cashflow["diff_quadratic_sum"].astype(float),11)
benford_cashflow_boundarys=benford_cashflow_boundarys[1:11]
#读取利润表数字的本福特统计结果跟理论结果的误差平方和
input_dir="D:/TMP/benford_income.csv"
benford_income=read_csv(input_dir)
benford_income=pd.DataFrame(benford_income[1:],columns=benford_income[0])
benford_income=benford_income.loc[:,["COMP_NAME","diff_quadratic_sum"]]
benford_income_boundarys=quantile(benford_income["diff_quadratic_sum"].astype(float),11)
benford_income_boundarys=benford_income_boundarys[1:11]

#将上述四列数据,合并到一个数据集
merge_set=pd.merge(benford_total, benford_balance, how='left', on=['COMP_NAME'])
merge_set=pd.merge(merge_set, benford_cashflow, how='left', on=['COMP_NAME'])
```

```
merge_set=pd.merge(merge_set,benford_income,how='left',on=['COMP_NAME'])
merge_set.columns=["COMP_NAME","counts","total","balance","cash_flow","income"]
#设置运算结果为浮点数,以便于保留小数位得分
merge_set.counts=merge_set.counts.astype(float)
merge_set.total=merge_set.total.astype(float)
merge_set.balance=merge_set.balance.astype(float)
merge_set.cash_flow=merge_set.cash_flow.astype(float)
merge_set.income=merge_set.income.astype(float)

score_list=[]
for i in range(len(merge_set.COMP_NAME)):
    print("counting object: "+str(i))
    weight_score_list=[]
    #赋权重
    for j in range(2,6):
        if merge_set.iloc[i,j]<benford_total_boundarys[0]:
            weight_score_list.append(0)
        elif merge_set.iloc[i,j]>=benford_total_boundarys[0] and merge_set.iloc[i,j]<benford_total_boundarys[1]:
            weight_score_list.append(1*1.1)
        elif merge_set.iloc[i,j]>=benford_total_boundarys[1] and merge_set.iloc[i,j]<benford_total_boundarys[2]:
            weight_score_list.append(2*1.2)
        elif merge_set.iloc[i,j]>=benford_total_boundarys[2] and merge_set.iloc[i,j]<benford_total_boundarys[3]:
            weight_score_list.append(3*1.3)
        elif merge_set.iloc[i,j]>=benford_total_boundarys[3] and merge_set.iloc[i,j]<benford_total_boundarys[4]:
            weight_score_list.append(4*1.4)
        elif merge_set.iloc[i,j]>=benford_total_boundarys[4] and merge_set.iloc[i,j]<benford_total_boundarys[5]:
            weight_score_list.append(5*1.5)
        elif merge_set.iloc[i,j]>=benford_total_boundarys[5] and merge_set.iloc[i,j]<benford_total_boundarys[6]:
            weight_score_list.append(6*1.6)
        elif merge_set.iloc[i,j]>=benford_total_boundarys[6] and merge_set.iloc[i,j]<benford_total_boundarys[7]:
            weight_score_list.append(7*1.7)
```

```
            elif merge_set.iloc[i,j]>=benford_total_boundarys[7] and
merge_set.iloc[i,j]<benford_total_boundarys[8]:
                weight_score_list.append(8 * 1.8)
            elif merge_set.iloc[i,j]>=benford_total_boundarys[8] and
merge_set.iloc[i,j]<benford_total_boundarys[9]:
                weight_score_list.append(9 * 1.9)
            else:
                weight_score_list.append(10 * 2)

    comprehensive_score=(20-sum(weight_score_list)/4)/2 #计算本福特得分
    #如果一家公司的数字太少,直接得-1分
    if merge_set.counts[i]<50:
        comprehensive_score=-1

    score_list.append(comprehensive_score)

merge_set["score"]=score_list
merge_set=merge_set.sort_values(by="score",axis=0,ascending=False)
#输出每家公司的最终本福特评分结果
output_dir="D:/TMP/benfordslaw_comprehensive_score.csv"
merge_set.to_csv(output_dir, index=False, sep=',')
```


根据本福特评分结果的输出文件,并结合已经违约的168只债券的发行主体,我们找出了其中有较多财务数据的34家已违约企业,它们的本福特得分分别如表6.2所示。

表6.2　部分已经违约债券发行主体的本福特得分

序号	公司名称	财报数据个数	总误差平方和	资产负债表误差平方和	现金流量表误差平方和	利润表误差平方和	本福特得分
1	江苏保千里视像科技集团股份有限公司	477	0.001 1	0.002 1	0.006	0.011 7	6.86
2	中国城市建设控股集团有限公司	462	0.002 4	0.002 2	0.004 5	0.016 7	6.49
3	广西有色金属集团有限公司	353	0.004 7	0.003 5	0.003 1	0.062 4	5.74
4	五洋建设集团股份有限公司	410	0.002 6	0.007 9	0.003 3	0.031 9	5.59
5	中科云网科技集团股份有限公司	442	0.001 5	0.010 6	0.005	0.010 7	5.46
6	神雾环保技术股份有限公司	497	0.003 4	0.000 5	0.017 1	0.014 7	5.24



续表

序号	公司名称	财报数据个数	总误差平方和	资产负债表误差平方和	现金流量表误差平方和	利润表误差平方和	本福特得分
7	河北省物流产业集团有限公司	425	0.002 9	0.008 2	0.003 6	0.016 1	5.19
8	内蒙古博源控股集团有限公司	615	0.002	0.005 3	0.007 9	0.018 3	5.04
9	保定天威集团有限公司	513	0.001 9	0.009 1	0.004 3	0.014 6	4.99
10	四川省煤炭产业集团有限责任公司	539	0.001 8	0.005	0.009 8	0.034 7	4.99
11	东北特殊钢集团有限责任公司	325	0.000 9	0.009 1	0.010 8	0.009 3	4.6
12	亿阳集团股份有限公司	560	0.003 6	0.012 7	0.003 4	0.033	4.54
13	协鑫集成科技股份有限公司	478	0.004 3	0.002 8	0.017 3	0.048	4.49
14	中煤集团山西华昱能源有限公司	507	0.004 8	0.006	0.007 4	0.037 2	4.24
15	中国中钢股份有限公司	244	0.005	0.004 6	0.010 9	0.021 5	4.19
16	南京雨润食品有限公司	369	0.003 5	0.005 3	0.010 4	0.055	4.16
17	丹东港集团有限公司	506	0.004 5	0.003 9	0.022 9	0.037 7	4.09
18	珠海中富实业股份有限公司	486	0.002 5	0.008 9	0.009 3	0.012 8	3.96
19	保定天威英利新能源有限公司	450	0.008 4	0.002 5	0.027 9	0.018 9	3.63
20	富贵鸟股份有限公司	426	0.008 1	0.006 3	0.009 9	0.042 5	3.38
21	神雾科技集团股份有限公司	499	0.006 4	0.004 5	0.022 8	0.069 2	3.3
22	华盛江泉集团有限公司	507	0.003 9	0.009 1	0.014 1	0.015	3.23
23	大连机床集团有限责任公司	403	0.005 7	0.007 8	0.033 5	0.022 5	3.04

续表

序号	公司名称	财报数据个数	总误差平方和	资产负债表误差平方和	现金流量表误差平方和	利润表误差平方和	本福特得分
24	淄博宏达矿业有限公司	296	0.007 1	0.018 2	0.010 9	0.008 4	2.78
25	山东山水水泥集团有限公司	308	0.005 4	0.009	0.025 8	0.036 7	2.73
26	春和集团有限公司	437	0.012 1	0.005	0.027 2	0.060 9	2.65
27	甘肃宏良皮业股份有限公司	347	0.009 1	0.009 4	0.009	0.028 9	2.46
28	神雾节能股份有限公司	409	0.007 4	0.025 3	0.008 7	0.012 2	2.44
29	四川圣达集团有限公司	329	0.006 1	0.014 1	0.012 4	0.026 9	2.39
30	河南佳源乳业股份有限公司	168	0.006 4	0.020 2	0.024 2	0.025 6	2.1
31	亿利资源集团有限公司	641	0.013 3	0.011 4	0.021 3	0.024 4	1.79
32	内蒙古奈伦集团股份有限公司	280	0.014 6	0.026 3	0.011 3	0.024 8	1.79
33	吉林粮食集团收储经销有限公司	134	0.010 1	0.012 6	0.057 7	0.098 9	1.79
34	亿阳信通股份有限公司	464	0.008 4	0.016 2	0.017 4	0.070 6	1.79

由表 6.2 的本福特得分结果可见，绝大多数的已违约企业处于“及格分”(6 分)以下。我们分别统计了这 34 家已经违约企业的本福特得分分布情况，如图 6.39 和图 6.40 所示。

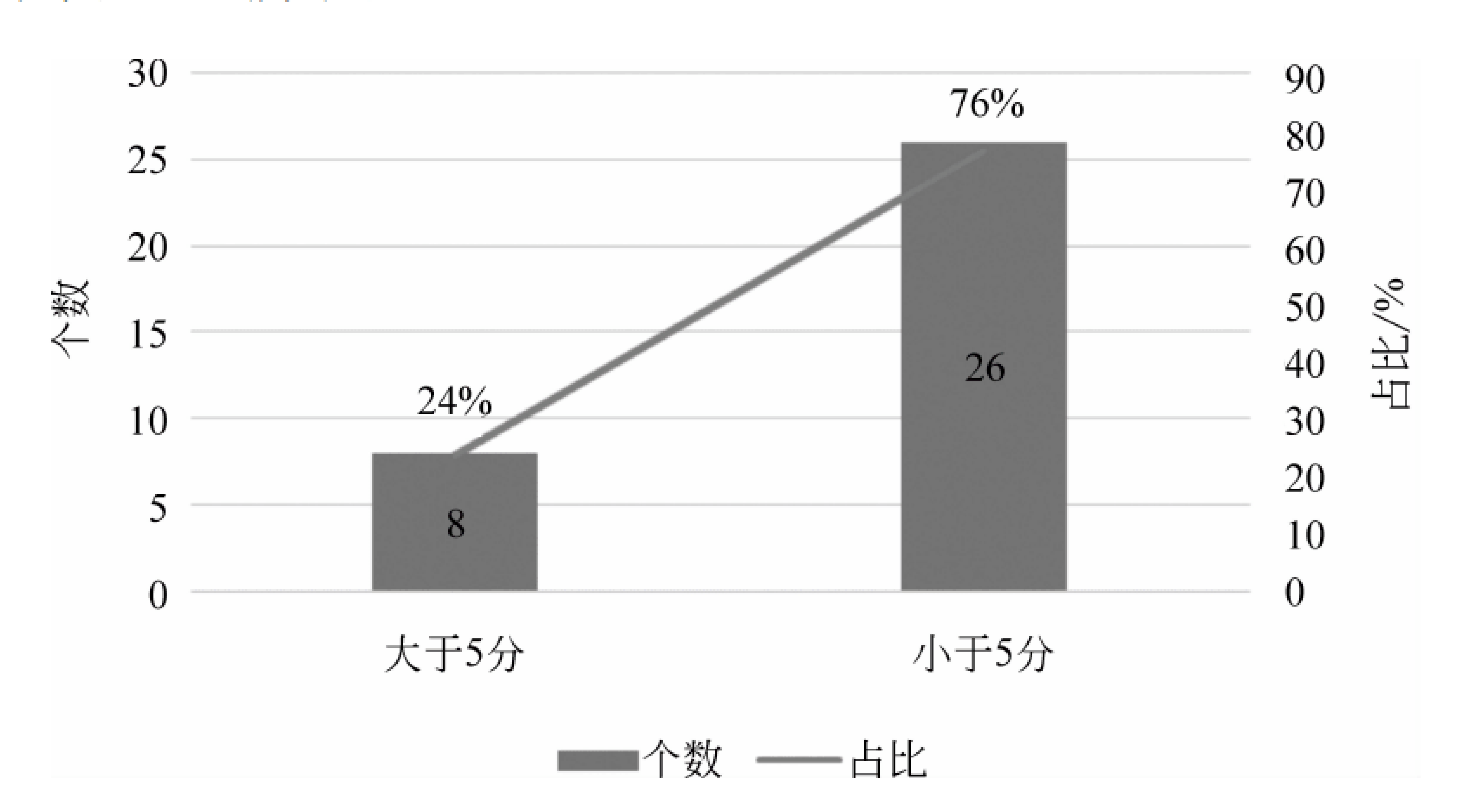


图 6.39　以 5 分为分割线的本福特财报粉饰图

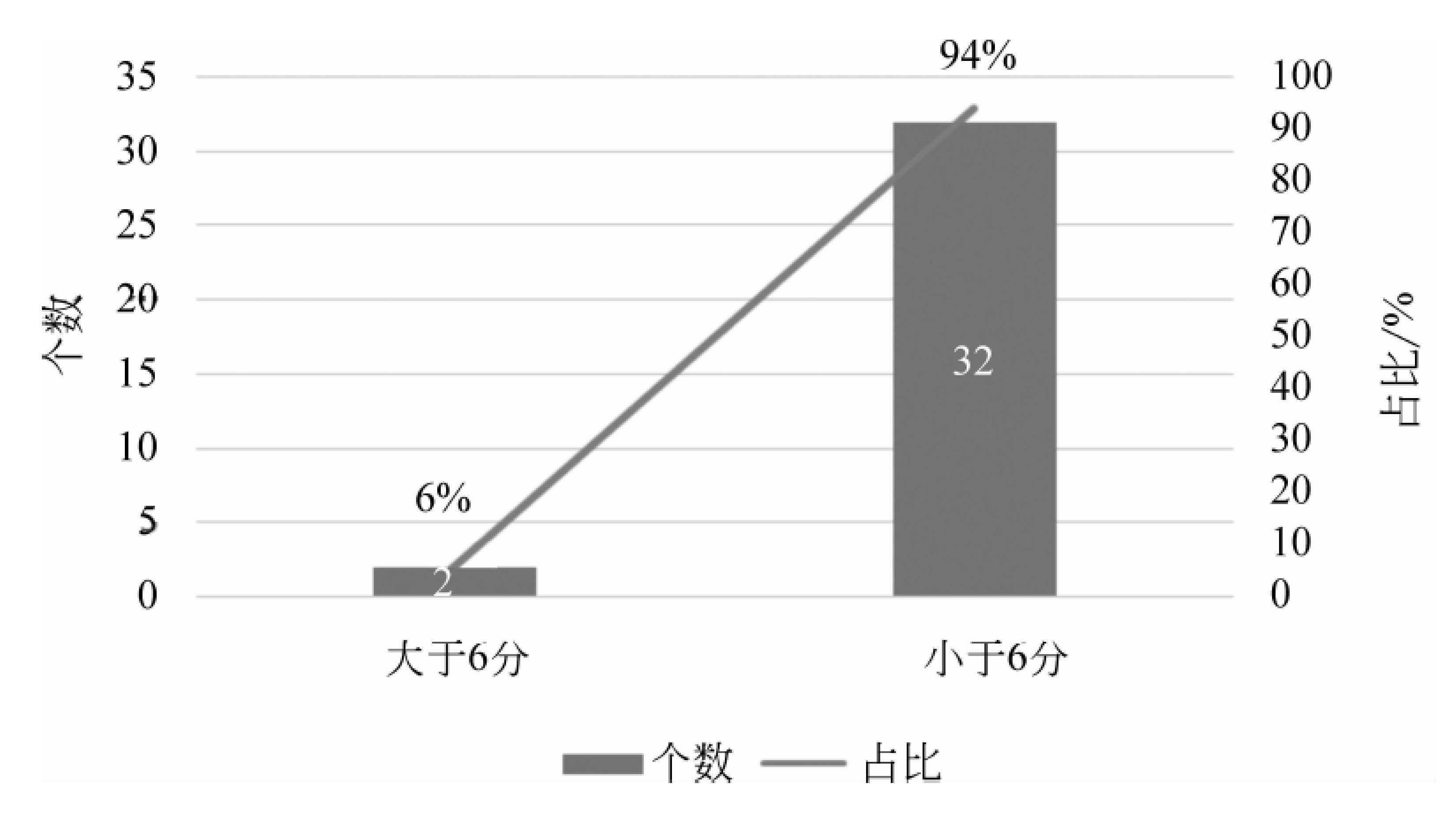


图 6.40　以 6 分为分割线的本福特财报粉饰图

第 7 章 基于场外数据的主体评级模型开发方法

由前几章的介绍可知，在国内资本市场的现状下，仅仅使用财务数据是很难评估信用债发行主体真实信用风险的。因此，根据笔者的深度研究结论，本章将重点介绍采用场外司法、招聘、股权出质、动产抵押等真实数据，并采用机器学习的方法来开发评估信用债发债主体真实信用风险的方法。此处，我们开发的信用风险评级模型不同于评分卡模型，如果读者想深入理解评分卡模型的开发方法，请阅读笔者的另外一本书《基于 R 语言的证券公司信用风险计量和管理》。在这本书中，笔者详细介绍了评分卡模型的开发方法。

7.1 有监督机器学习方法开发模型及 Python 源代码

本节重点介绍采用有监督机器学习技术开发评级模型的方法，并提供详细的 Python 源代码和注释说明。该类机器学习技术有别于评分卡模型的开发方法，主要体现在两者基本原理的不同。

有监督机器学习技术既可用于对信用债发行主体违约概率的预测，也可用于分类。如果用于预测，则可将非线性表达式表示的模型转换为评分卡模型；如果用于分类，则只能使用样本集训练分类器，并使用该分类器对测试集的样本进行分类运算，输出的结果也只有“违约”或“正常”这样的类别，无法计算违约概率，也无法转换为评分卡模型。使用有监督机器学习技术开发的模型，一般都是抽象的、非线性多项式表示的模型，这对于非计算机专业出身的信用债投资者来说，理解非常困难；且对于存在疑问的输出结果，通常较难找到直接的原因。

评分卡模型可以直观地给出每个信用债发行主体的分数，并能够详细追踪每个指标的得分及该指标对总得分的贡献。在开发原理方面，评分卡模型的开发需要首先对每个指标进行证据权重(WOE)转换，然后使用逻辑回归函数拟合，并设定初始分数和比率翻番的分数(PDO)即可生成评分卡模型。

本节采用的数据集来自笔者正在做的一个项目，因涉及大量的商业信息，因此做了脱敏处理。该数据集共计包含 8 155 个样本，其中违约样本 160 个，入模指标 18 个，违约状态指标 1 个。模型的详细 Python 代码，如下所示。

```
#导入模型开发和检验所需的 Python 包
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
plt.style.use('ggplot')  #风格设置近似 R 这种的 ggplot 库
import seaborn as sns
#设置输出图片保存格式和字体
plt.rcParams['font.sans-serif']=['SimHei'] #指定默认字体
plt.rcParams['axes.unicode_minus']=False #解决保存图像是负号'-'显示为方块的问题
#忽略弹出的 warnings
import warnings
warnings.filterwarnings('ignore')
#导入机器学习相关 Python 包
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_predict
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.cross_validation import train_test_split
from imblearn.over_sampling import SMOTE #导入 SMOTE 算法模块
from sklearn.preprocessing import StandardScaler
#读入数据集,该数据集已做脱敏和缺失值填充处理
df=pd.read_excel("chapter7_data.xlsx")
df.head()#查看前 5 行,如表 7.1 所示
```

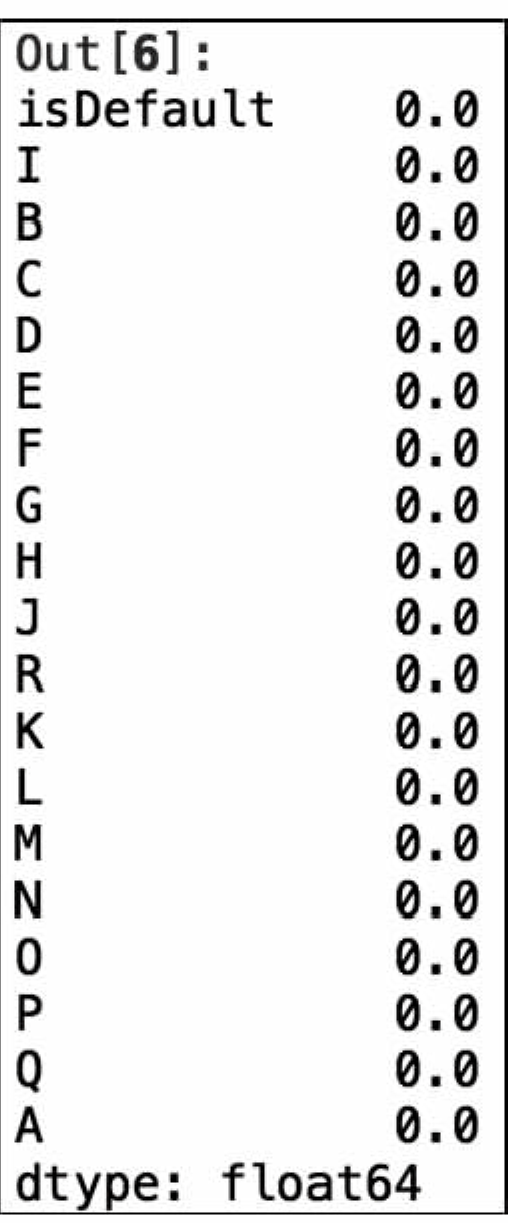
表 7.1　数据集的前五行数据

A	B	C	D	E	F	G	H	I	J
－11	－13	9.2	－89	－12	2.6	－22	－20	－29	－20
－11	－13	－26	－89	－12	2.6	－22	－20	－29	－20
－11	－13	9.2	－89	－12	2.6	1.8	1.3	－16	4.9
－11	－13	－3.9	－89	7.7	2.6	1.8	1.3	4.7	－20
－11	－13	－3.9	－40	7.7	－15	－22	－20	－29	－20

K	L	M	N	O	P	Q	R	isDefault
－3.8	4.6	－17	－5.5	－13	－6.1	2.2	－2.4	1
－3.8	4.6	7	－4.5	6.8	－6.1	2.2	－2.4	1
－3.8	－15	－42	－4.5	－13	－6.1	－12	－2.4	1
－3.8	－15	－42	－4.5	－13	－6.1	2.2	－11	1
－7.2	4.6	－17	－5.5	－13	－6.1	2.2	－2.4	1

```
#查看数据集缺失值情况
df.isnull().sum(axis=0).sort_values(ascending=False)/float(len
(df)) #缺失值统计结果,如图 7.1 所示

#查看数据集的维数和数据集的列名
df.shape #19 列 8 155 个样本
df.columns #显示数据集的列名
cols_of_feature=df.columns[:-1] #选择特征变量
#目标变量分布可视化
fig, axs=plt.subplots(1,2,figsize=(14,7))
sns.countplot(x='isDefault',data=df,ax=axs
[0])
axs[0].set_title("Frequency of each Target")
df['isDefault'].value_counts().plot(x=None,
y=None, kind='pie', ax=axs[1], autopct=
'%1.2f%%')
axs[1].set_title("Percentage of each Target")
fig.savefig("Normal VS Default")
plt.show() #绘制数据集目标变量分布图,如图 7.2 所示
```



```
Out[6]:
isDefault    0.0
I            0.0
B            0.0
C            0.0
D            0.0
E            0.0
F            0.0
G            0.0
H            0.0
J            0.0
R            0.0
K            0.0
L            0.0
M            0.0
N            0.0
O            0.0
P            0.0
Q            0.0
A            0.0
dtype: float64
```

图 7.1　缺失值统计图

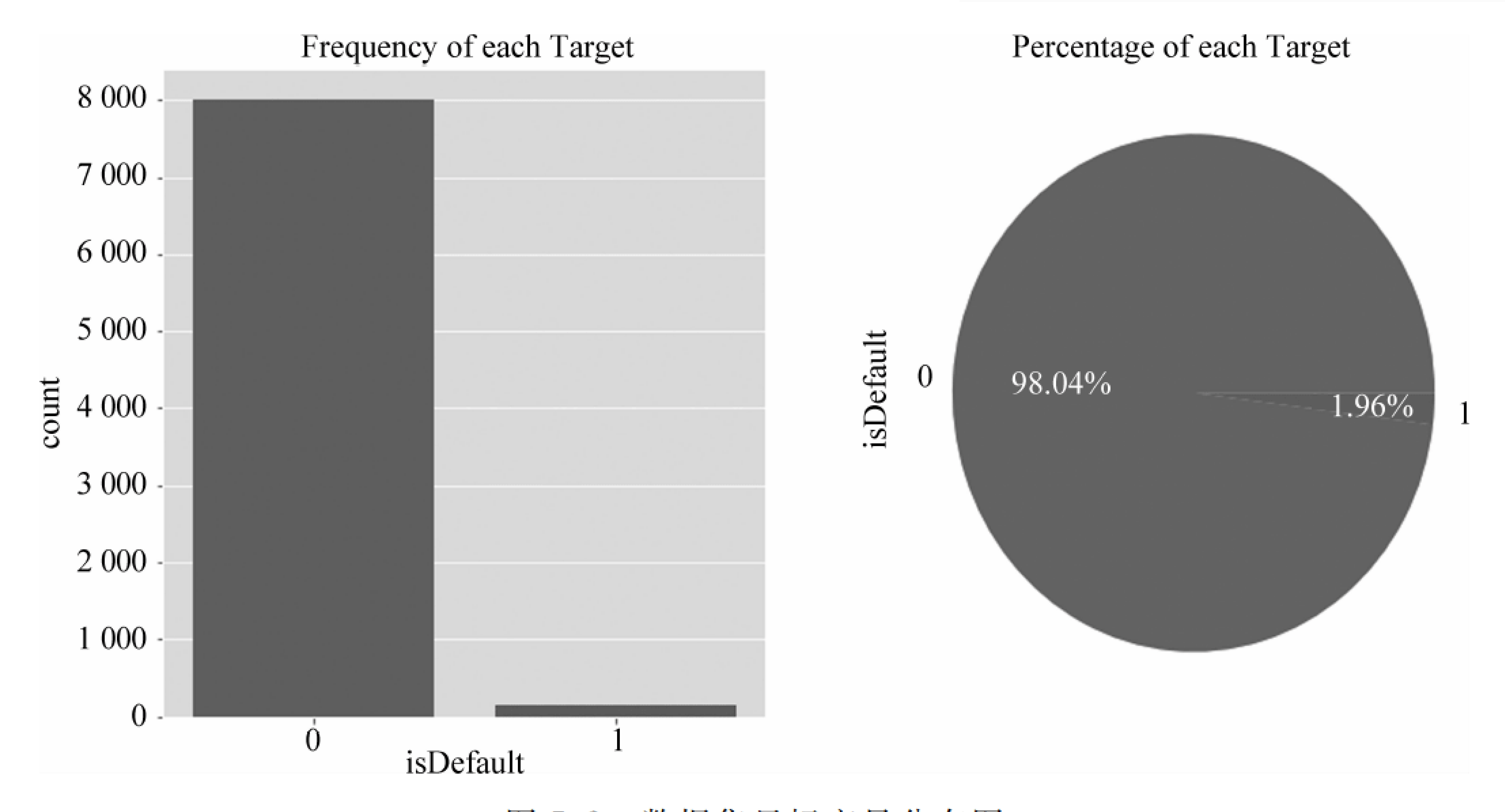


图 7.2　数据集目标变量分布图

```
#可见,数据集中正常样本和违约样本的占比极度不均衡。正常样本占比 98.04%,违
约样本占比 1.96%。
#查看数据集中正常样本和违约样本的个数
df['isDefault'].value_counts()#7 995 个正常样本,160 个违约样本
#在不同目标变量下,对比指标的相关性,并绘制相关性图谱
XDefault=df.loc[df["isDefault"]==1]
```

```
XnonDefault=df.loc[df["isDefault"]==0]
correlationNonDefault=XnonDefault.loc[:, df.columns ! ='isDefault'].
corr()
mask=np.zeros_like(correlationNonDefault)
indices=np.triu_indices_from(correlationNonDefault)
mask[indices]=True
grid_kws={"width_ratios": (.9, .9, .05), "wspace": 0.2}
f, (ax1, ax2, cbar_ax) = plt.subplots(1, 3, gridspec_kw=grid_kws,
figsize=(14, 9))
cmap=sns.diverging_palette(220, 8, as_cmap=True)
ax1= sns.heatmap(correlationNonDefault, ax= ax1, vmin= -1, vmax= 1,
cmap=cmap, square=False, linewidths=0.5, mask=mask, cbar=False)
ax1.set_xticklabels(ax1.get_xticklabels(), size=16)
ax1.set_yticklabels(ax1.get_yticklabels(), size=16)
ax1.set_title('Non Default', size=20)
correlationDefault=XDefault.loc[:, df.columns ! ='isDefault'].corr()
ax2= sns.heatmap(correlationDefault, vmin= -1, vmax= 1, cmap= cmap,
ax= ax2, square = False, linewidths = 0. 5, mask = mask, yticklabels =
False, cbar_ax=cbar_ax,cbar_kws={'orientation': 'vertical','ticks':
[-1, -0.5, 0, 0.5, 1]})
ax2.set_xticklabels(ax2.get_xticklabels(), size=16)
ax2.set_title('Default', size=20)
cbar_ax.set_yticklabels(cbar_ax.get_yticklabels(), size=14)
plt.savefig('Feature Correlation', bbox_inches='tight') #输出指标的
相关性图谱,如图 7.3 所示
```

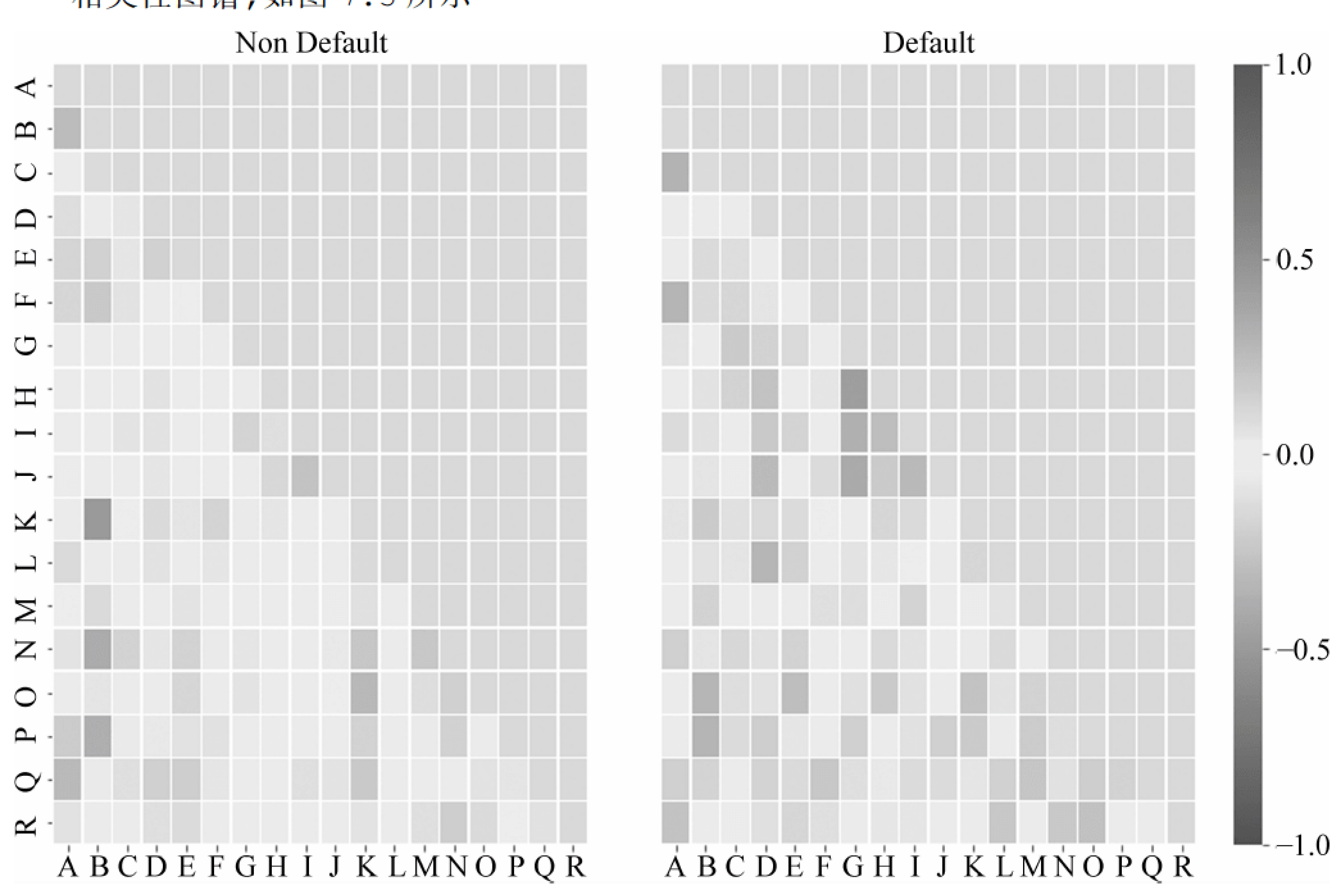


图 7.3　不同目标变量的指标相关性图谱

```
#通过随机森林算法,计算特征重要性,并绘制特征重要性累计分布图
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=10,random_state=123)#构建
分类随机森林分类器
clf.fit(x_val[cols_of_feature], y_val) #对自变量和因变量进行拟合
cols_of_feature, clf.feature_importances_
for feature in zip(cols_of_feature, clf.feature_importances_):
    print(feature)
#绘制特征重要新累计分布图,如图7.4所示
```

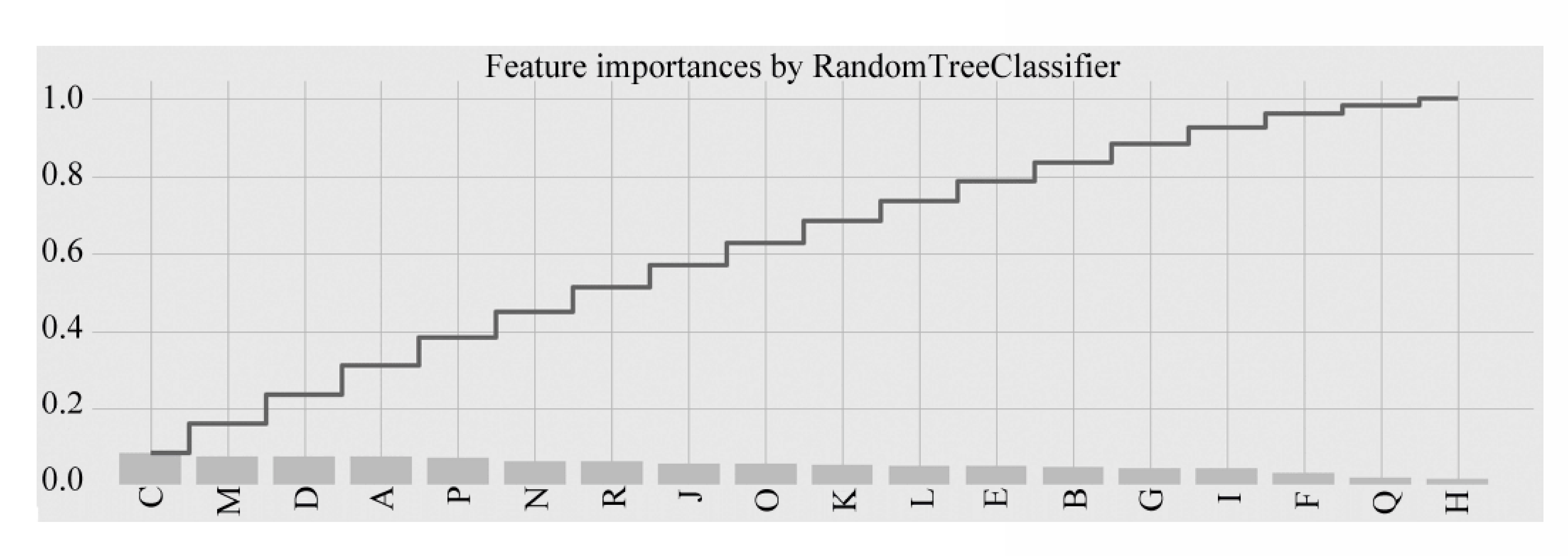


图7.4　特征重要新累计分布图


```
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize']=(12,6)
#根据输出的各指标重要性排序结果,剔除重要性最弱的两个变量Q、H
cols_to_drop=['Q','H']
col_val=df[cols_of_feature].columns.drop(cols_to_drop)

##训练模型前的准备工作
#构建自变量和因变量
X=df[col_val]
y=df["isDefault"]

#将模型划分为训练集和测试集
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.3,
random_state=0) #

#处理样本不平衡,只对训练集作处理
from imblearn.over_sampling import SMOTE #导入SMOTE算法模块
sm=SMOTE(random_state=42)      #处理过采样的方法
X_train, y_train=sm.fit_sample(X_train, y_train)
print('通过SMOTE方法平衡正负样本后')
```

```
n_sample=y_train.shape[0]
n_pos_sample=y_train[y_train==0].shape[0]
n_neg_sample=y_train[y_train==1].shape[0]
print('样本个数:{}; 正样本占{:.2%}; 负样本占{:.2%}'.format(n_sample,n_
pos_sample / n_sample,n_neg_sample / n_sample))

#通过 SMOTE 方法平衡正负样本后
#样本个数:11 210; 正样本占 50.00%; 负样本占 50.00%

#模型训练和比较
#此处采用多种机器学习方法来训练模型,包括 AdaBoost、Bagging、ExtraTrees、
GradientBoosting、RandomForest、GaussianProcess、SVM 等
from sklearn import svm, model _ selection, tree, linear _ model,
neighbors, naive_bayes, ensemble, discriminant_analysis, gaussian_
process
from sklearn.metrics import mean_squared_error, confusion_matrix,
precision_score, recall_score, auc,roc_curve
import seaborn as sns
#汇总各机器学习方法,以便于遍历训练
MLA=[
    #Ensemble Methods
    ensemble.AdaBoostClassifier(),
    ensemble.BaggingClassifier(),
    ensemble.ExtraTreesClassifier(),
    ensemble.GradientBoostingClassifier(),
    ensemble.RandomForestClassifier(),
    #Gaussian Processes
    gaussian_process.GaussianProcessClassifier(),
    #GLM
    linear_model.LogisticRegressionCV(),
    linear_model.PassiveAggressiveClassifier(),
    linear_model.RidgeClassifierCV(),
    linear_model.SGDClassifier(),
    linear_model.Perceptron(),
    #Navies Bayes
    naive_bayes.BernoulliNB(),
    naive_bayes.GaussianNB(),
    #Nearest Neighbor
    neighbors.KNeighborsClassifier(),
    #SVM
    svm.SVC(probability=True),
    svm.NuSVC(probability=True),
```

```
svm.LinearSVC(),
#Trees
tree.DecisionTreeClassifier(),
]

#拟合模型
MLA_columns=[]
MLA_compare=pd.DataFrame(columns=MLA_columns)
row_index=0
for alg in MLA:
    predicted=alg.fit(X_train, y_train).predict(X_test)
    fp, tp, th=roc_curve(y_test, predicted)
    MLA_name=alg.__class__.__name__
    MLA_compare.loc[row_index,'MLA Name']=MLA_name
    MLA_compare.loc[row_index, 'MLA Train Accuracy']=round(alg.
score(X_train, y_train), 4)
    MLA_compare.loc[row_index, 'MLA Test Accuracy']=round(alg.
score(X_test, y_test), 4)
    MLA_compare.loc[row_index, 'MLA Precission']=precision_
score(y_test, predicted)
    MLA_compare.loc[row_index, 'MLA Recall']=recall_score(y_
test, predicted)
    MLA_compare.loc[row_index, 'MLA AUC']=auc(fp, tp)
    row_index+ =1

MLA_compare.sort_values(by= ['MLA Test Accuracy'], ascending=
False, inplace=True)
MLA_compare
#输出各机器学习算法检验比较结果,如表 7.2 所示
```

表 7.2　各机器学习算法检验比较　　%

MLA Name	MLA Train Accuracy	MLA Test Accuracy	MLA Precission	MLA Recall	MLA AUC
GaussianNB	81.5	89.5	15.2	77.2	0.83
PassiveAggressiveClassifier	84.9	84.3	11.2	82.5	0.83
BemoulliNB	84.3	87.1	12.9	78.9	0.83
LogisticRegressionCV	86.8	86.4	12.3	78.9	0.83
RidgeClassifierCV	86.5	85.7	11.8	78.9	0.82
LinearSVC	85.3	81.0	9.2	80.7	0.81

续表

MLA Name	MLA Train Accuracy	MLA Test Accuracy	MLA Precission	MLA Recall	MLA AUC
KNeighborsClassifier	98.2	94.9	25.2	61.4	0.79
SGDClassifier	79.1	79.4	7.6	70.2	0.75
AdaBoostClassifier	98.0	96.4	32.1	47.4	0.72
GradientBoostingClassifier	99.3	97.8	55.9	33.3	0.66
Perceptron	65.8	59.2	3.7	66.7	0.63
GaussianProcessClassifier	100.0	96.2	21.5	24.6	0.61
ExtraTreesClassifier	100.0	97.8	60.0	21.1	0.60
DecisionTreeClassifier	100.0	96.3	20.0	19.3	0.59
BaggingClassifier	99.9	97.4	36.0	15.8	0.58
RandomForestClassifier	99.9	97.7	53.8	12.3	0.56
NuSVC	100.0	97.6	0.0	0.0	0.50
SVC	100.0	97.6	0.0	0.0	0.50

```
#比较训练集上不同模型的准确率(Accuracy)
plt.subplots(figsize=(15,6))
sns.barplot(x="MLA Name", y="MLA Train Accuracy",data=MLA_compare,palette='hot',edgecolor=sns.color_palette('dark',7))
plt.xticks(rotation=90)
plt.title('MLA Train Accuracy Comparison')
plt.show()
#绘制各机器学习算法训练集检验准确率比较图,如图7.5所示
#比较测试集上不同模型的准确率
plt.subplots(figsize=(15,6))
sns.barplot(x="MLA Name", y="MLA Test Accuracy",data=MLA_compare,palette='hot',edgecolor=sns.color_palette('dark',7))
plt.xticks(rotation=90)
plt.title('MLA Test Accuracy Comparison')
plt.show()
#绘制各机器学习算法测试集检验准确率(Accuracy)比较图,如图7.6所示
#比较不同模型的精准率
plt.subplots(figsize=(15,6))
sns.barplot(x="MLA Name", y="MLA Precission",data=MLA_compare,palette='hot',edgecolor=sns.color_palette('dark',7))
plt.xticks(rotation=90)
```

```
plt.title('MLA Precission Comparison')
plt.show()
#绘制各机器学习算法检验精准率(Precision)比较图,如图 7.7 所示
```

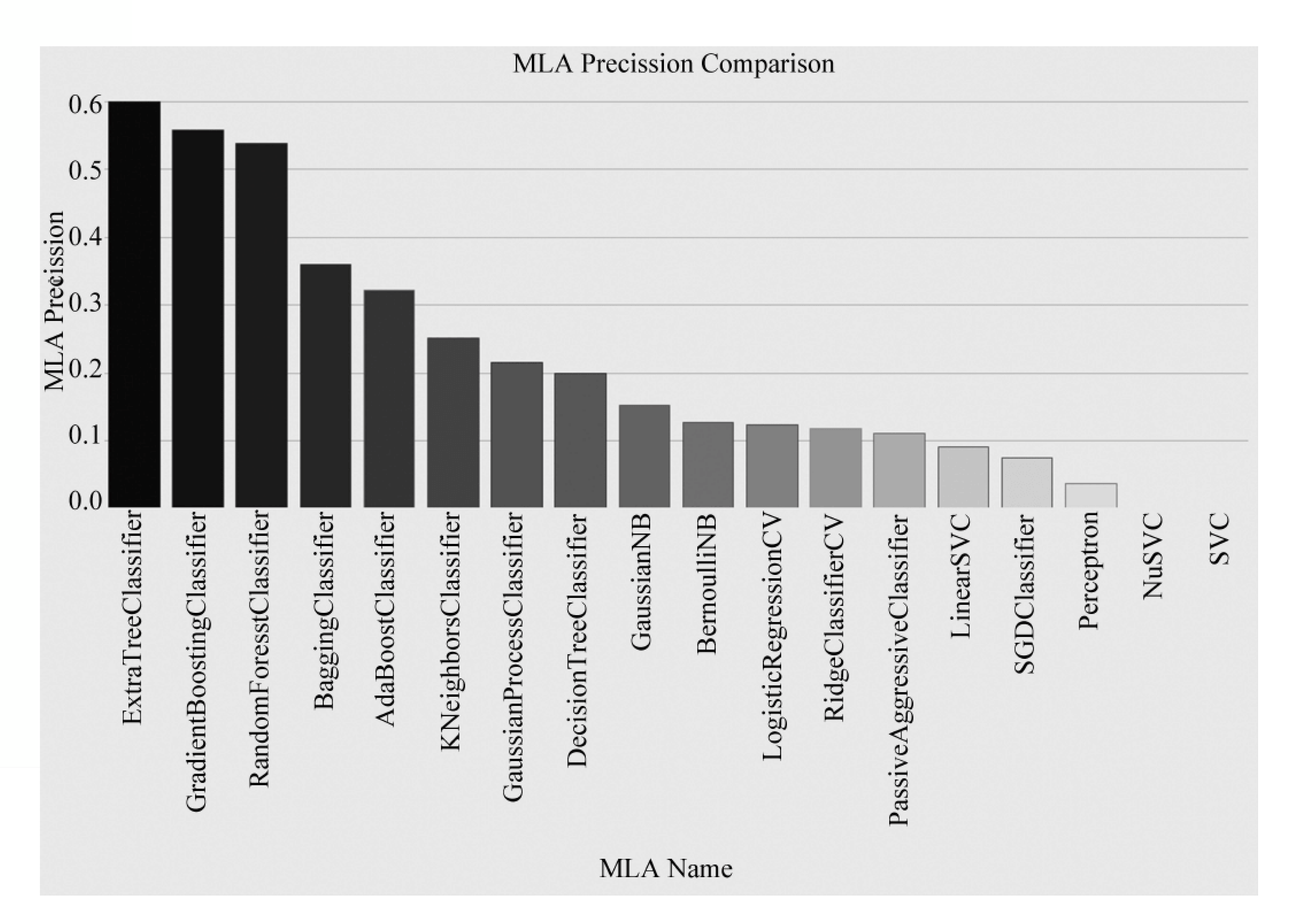


图 7.7　绘制各机器学习算法检验准确率比较图

```
#比较不同模型的召回率
plt.subplots(figsize=(15,6))
sns.barplot(x="MLA Name", y="MLA Recall",data=MLA_compare,
palette='hot',edgecolor=sns.color_palette('dark',7))
plt.xticks(rotation=90)
plt.title('MLA Recall Comparison')
plt.show()
#绘制各机器学习算法召回率(Recall)比较图,如图 7.8 所示
#比较不同模型的 AUC
plt.subplots(figsize=(15,6))
sns.barplot(x="MLA AUC", y="MLA Name", data=MLA_compare, palette=
'hot', edgecolor=sns.color_palette('dark',7))
plt.xticks(rotation=90)
plt.title('MLA AUC Comparison')
plt.show()
#绘制各机器学习算法 AUC 比较图,如图 7.9 所示
#绘制 ROC 比较曲线
```

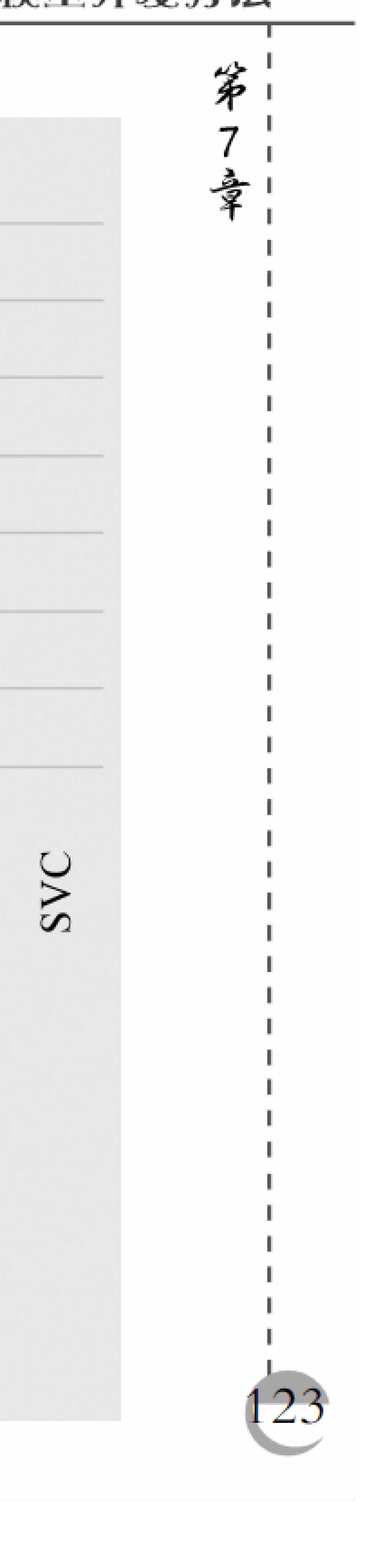


图 7.8　绘制各机器学习算法训练召回率比较图

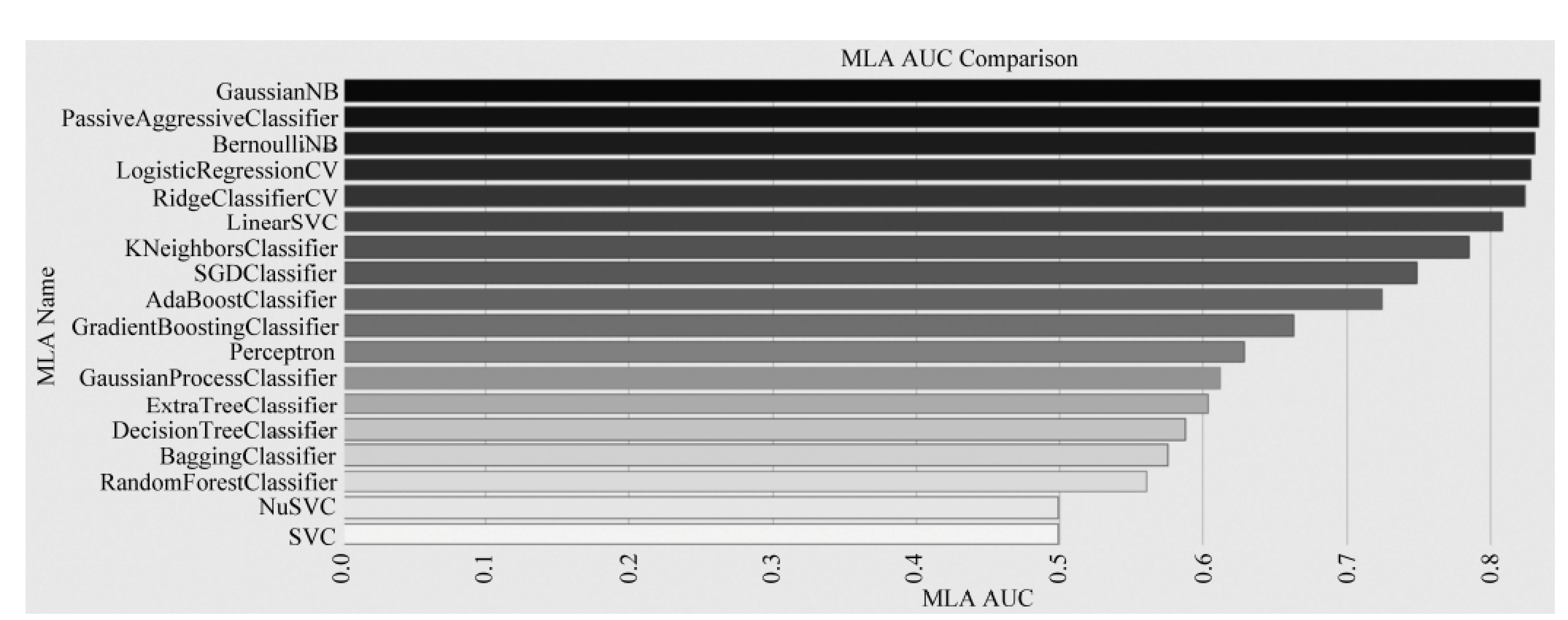


图 7.9　绘制各机器学习算法 AUC 比较图

```
index=1
for alg in MLA:
    predicted=alg.fit(X_train, y_train).predict(X_test)
    fp, tp, th=roc_curve(y_test, predicted)
    roc_auc_mla=auc(fp, tp)
    MLA_name=alg.__class__.__name__
    plt.plot(fp, tp, lw=2, alpha=0.3, label='ROC %s (AUC=%0.2f)'
%(MLA_name, roc_auc_mla))
```

```
    index+ =1

plt.subplots(figsize=(15,6)) #test
plt.title('ROC Curve comparison')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.plot([0,1],[0,1],'r--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

#绘制各机器学习算法的 ROC 曲线比较图,如图 7.10 所示

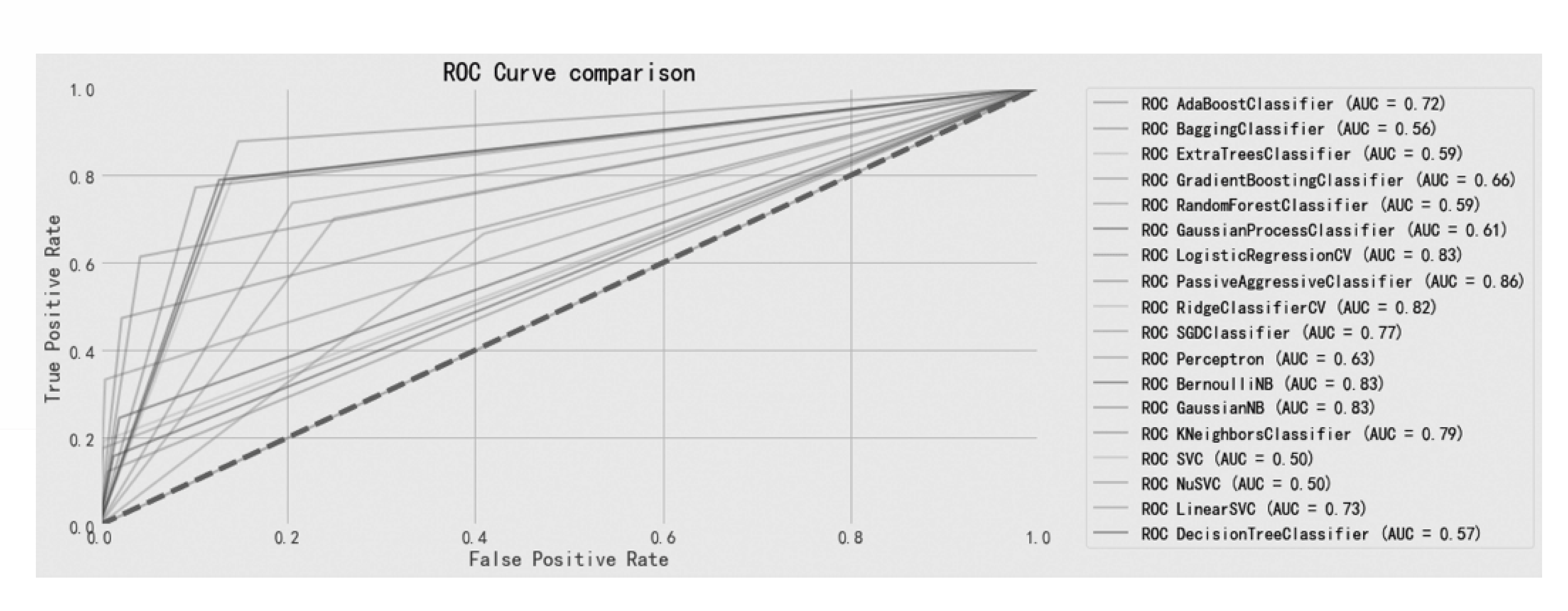


图 7.10 绘制各机器学习算法的 ROC 曲线比较图

#由以上各机器学习算法的检验结果可知,逻辑回归算法对本数据集的分类效果最好,因此我们选择逻辑回归算法进行优化调参。

#模型调参

```
from sklearn.model_selection import GridSearchCV
param_grid={'C': [0.01,0.1, 1, 10, 100, 1000,],'penalty': [ 'l1', 'l2']}
clf_logreg_GridSearch=GridSearchCV(LogisticRegression(), param_grid, scoring='roc_auc', cv=10) #确定模型 LogisticRegression 和参数组合 param_grid ,cv 指定 5 折
clf_logreg_GridSearch.fit(X_train, y_train) #使用训练集学习算法
clf_logreg_GridSearch.best_estimator_ #输出调参后的最优参数
clf_logreg_GridSearch.best_params_ # 输出结果表明,C 为 0.01, penalty 为 L1 时 为模型最优参数
clf_logreg_GridSearch.best_score_ #输出最优分数
```

#选取 AUC 分数最高的参数组合作为最终的模型参数

```
clf_logreg=clf_logreg_GridSearch.best_estimator_
from sklearn.metrics import auc
from sklearn.metrics import roc_curve

y_pred_test_prob=clf_logreg.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds=roc_curve(y_test,y_pred_test_prob)
roc_auc=auc(fpr,tpr)
roc_auc #最优模型的 AUC 为 0.92

#绘制 ROC 曲线
plt.figure(figsize=(14,7))
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b',label='AUC=%0.5f'%roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.0])
plt.ylim([-0.1,1.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.savefig('Receiver Operating Characteristic', bbox_inches=
'tight')
plt.show()
#绘制最优模型的 ROC 曲线,如图 7.11 所示
```

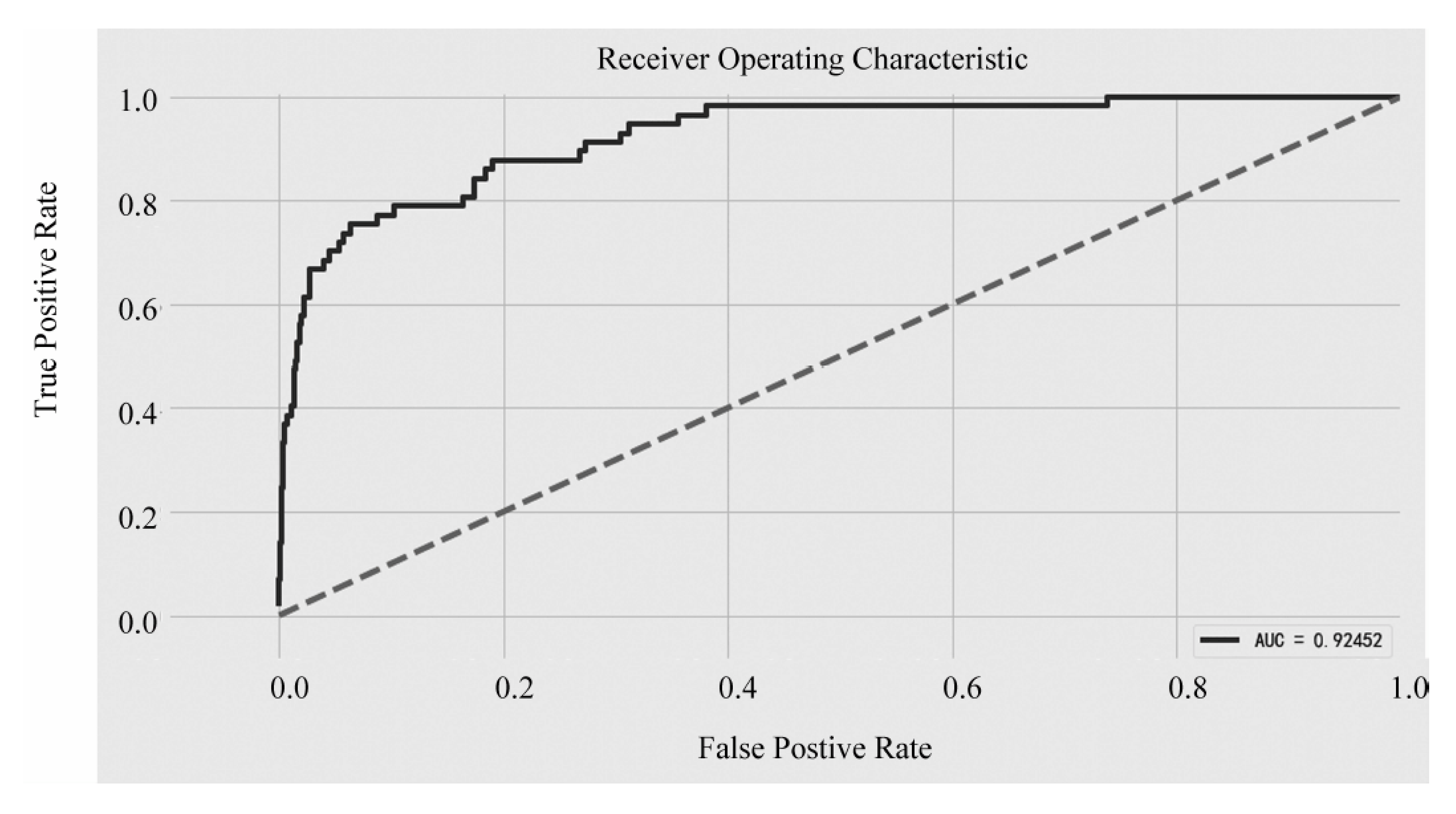


图 7.11　绘制最优模型的 ROC 曲线

很多金融机构并非按照默认的违约临界点来判定是否违约,或作为是否投

资的判定标准。此处，我们介绍一种通过自主设定违约临界点阈值的方法来判定模型的检验效果，并输出每个阈值的混淆矩阵。

```
import itertools
#绘制混淆矩阵函数
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks=np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=0)
    plt.yticks(tick_marks, classes)
    thresh=cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape
[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j]>thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

    return


y_pred_proba=clf_logreg.predict_proba(X_test)  #predict_prob 获得概
率值
thresholds=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]  #设定不同阈值

plt.figure(figsize=(15,10))
#绘制不同阈值下的混淆矩阵，如图 7.12 所示
j=1
for i in thresholds:
    y_test_predictions_high_recall=y_pred_proba[:,1]>i #预测出来的概
率值是否大于阈值

    plt.subplot(3,3,j)
    j + =1

    #Compute confusion matrix
```

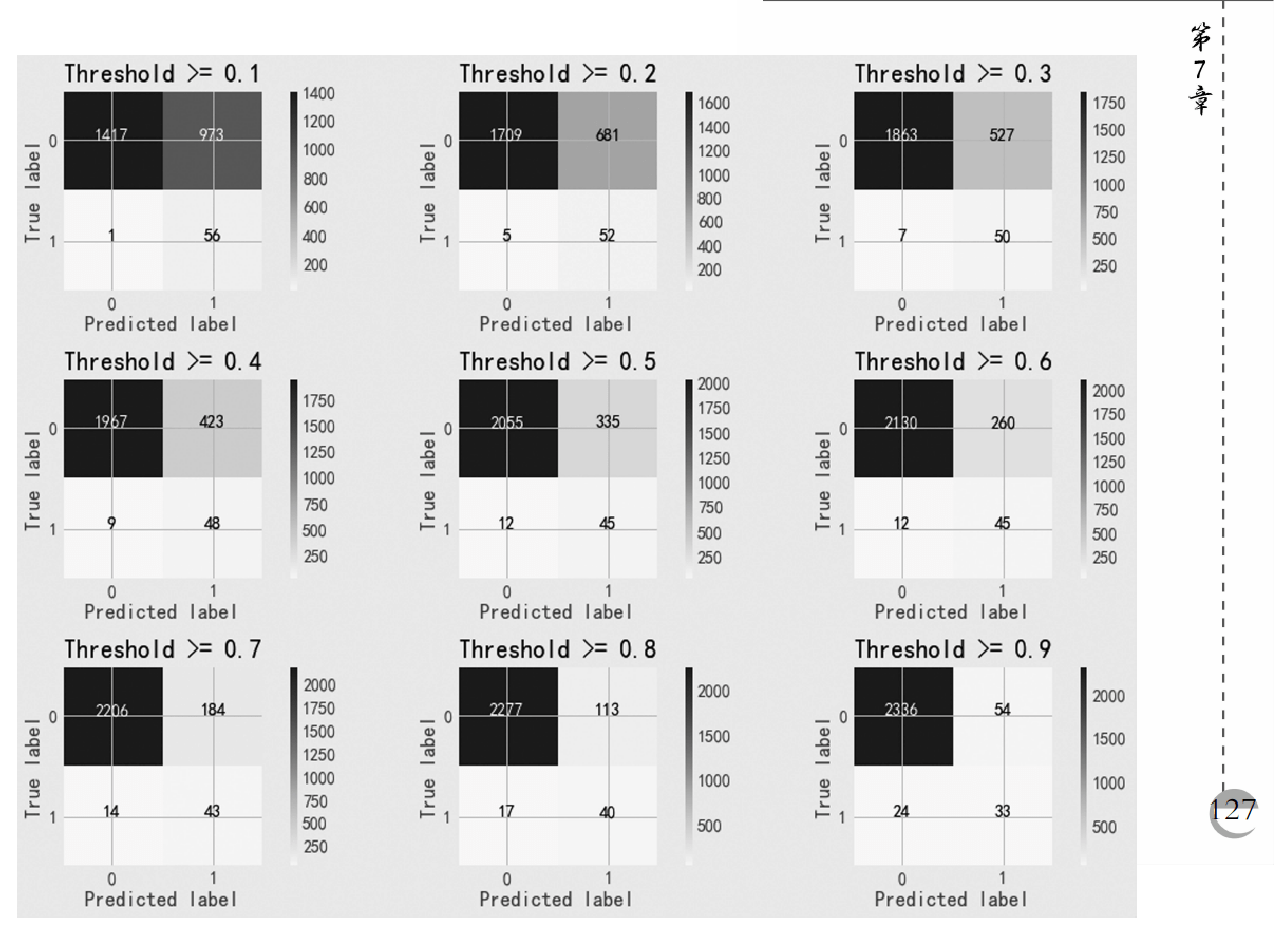


图 7.12　不同阈值下的混淆矩阵

```
    cnf_matrix=confusion_matrix(y_test, y_test_predictions_high_
recall)
    np.set_printoptions(precision=2)

    print("Recall metric in the testing dataset: ", cnf_matrix[1,1]/
(cnf_matrix[1,0]+ cnf_matrix[1,1]))

    #Plot non-normalized confusion matrix
    class_names=[0,1]
    plot_confusion_matrix(cnf_matrix
                          , classes=class_names
                          , title='Threshold >=%s'%i)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

from itertools import cycle
from sklearn.metrics import precision_recall_curve

thresholds=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
```

```
colors=cycle(['navy', 'turquoise', 'darkorange', 'cornflowerblue',
'teal', 'red', 'yellow', 'green', 'blue','black'])
plt.figure(figsize=(12,7))
#绘制不同阈值下的 ROC 曲线,如图 7.13 所示
```

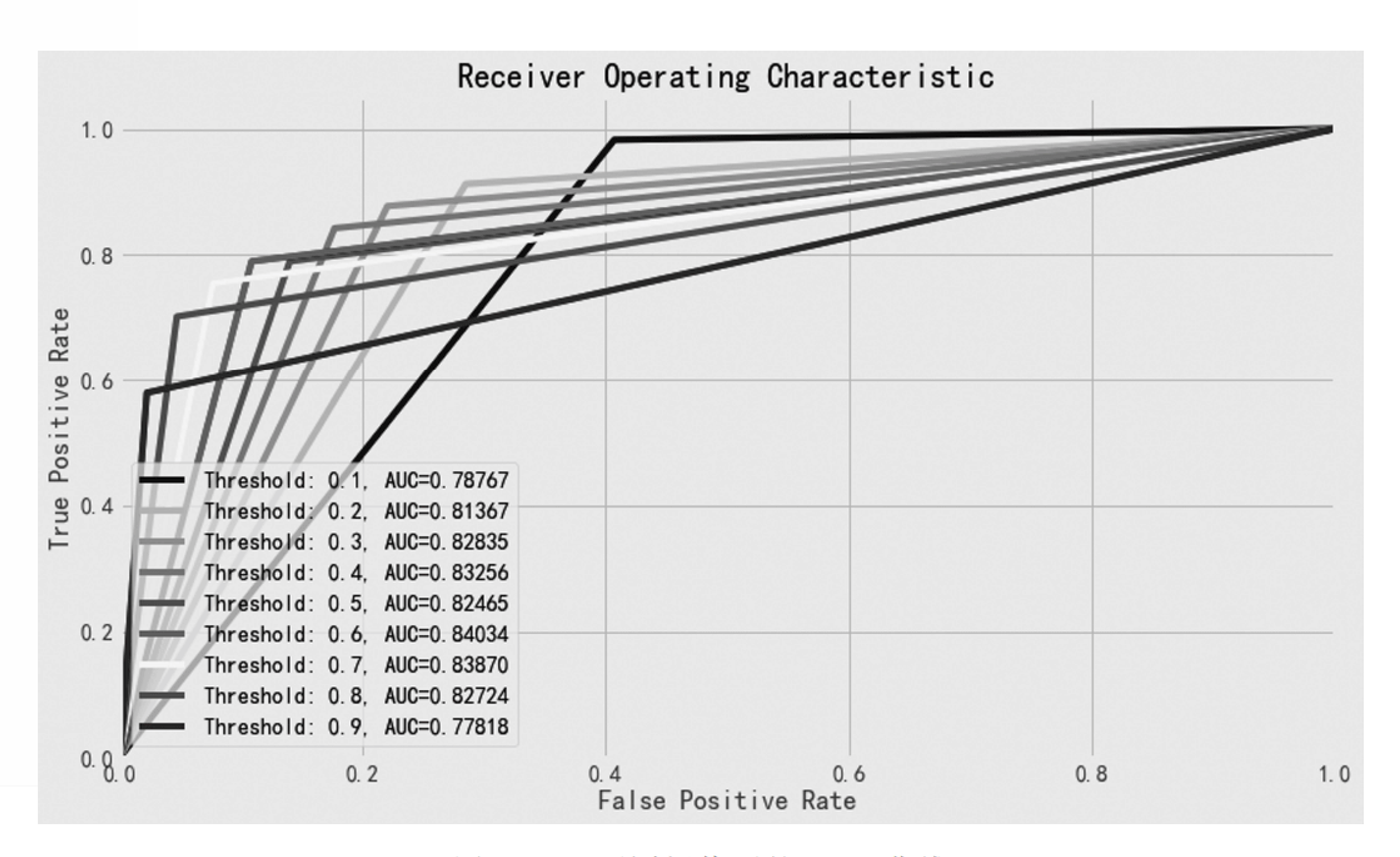


图 7.13　不同阈值下的 ROC 曲线

```
j=1
for i,color in zip(thresholds,colors):
    y_test_predictions_prob=y_pred_proba[:,1]>i #预测出来的概率值是否大于阈值
    fpr, tpr, thresholds=roc_curve(y_test,y_test_predictions_prob)
    roc_auc=auc(fpr, tpr)
    #绘制 ROC 曲线
    plt.plot(fpr, tpr, color=color,
                label='Threshold: %s, AUC=%0.5f' %(i , roc_auc))
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower left")
```

本节主要介绍了基于场外脱敏数据的有监督机器学习模型开发方法，并给出了多种机器学习技术的模型检验结果比较。需要指出的是获取稳定的场外数据源并提取相关入模指标是模型开发的关键，由于涉及商业机密，笔者对数据做

了脱敏处理,详细的处理方法请读者自行实践。

7.2 深度学习开发模型及 Python 源代码

由本书 2.2 节使用深度学习做分类算法的案例可知,使用基于 TensorFlow 后台的 Keras 库做深度学习非常方便,只需要几行代码就可以构建深度学习训练模型。

本节主要向读者详述使用脱敏数据集做分类任务的深度学习模型构建方法,相关 Python 代码如下所示。

```
#加载所需 Python 包
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense,Activation
from keras.optimizers import RMSprop
from imblearn.combine import SMOTEENN
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

#加载数据集
input_set=pd.read_excel("chapter7_data.xlsx")
#将模型划分为训练集和测试集
X, Y=np.array(input_set.iloc[:,:-1]), np.array(input_set.iloc[:,-1])
X_train, X_test, Y_train, Y_test=train_test_split(X, Y, test_size=0.3,
random_state=0) #random_state=0 每次切分的数据都一样

#过采样违约样本,解决样本不均衡问题
sm=SMOTE(kind='borderline2')
X_resampled, Y_resampled=sm.fit_sample(X_train, Y_train)

y_train=np_utils.to_categorical(Y_resampled)
y_test=np_utils.to_categorical(Y_test)
#构建深度学习模型
model=Sequential([
        Dense(7, input_dim=18),
        Activation("relu"),
        Dense(2),
        Activation("softmax")
```

```
        ])
#深度学习优化
rmsprop=RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
#编译深度学习模型
model.compile(
        optimizer=rmsprop,
        loss="categorical_crossentropy",
        metrics=["accuracy"]
        )
#训练深度学习模型,共计训练20次
train_process=model.fit(X_resampled,y_train,nb_epoch=20,batch_
size=100)
#使用训练好的模型,评估测试集
loss,accuracy=model.evaluate(X_test,y_test)
#输出训练结果
print("\ntest loss:",loss)
print("\ntest accuracy:",accuracy)
print("\nAccuracy for mark all not default: ", (len(Y_test)-sum(Y_
test))/len(Y_test))
#使用训练得到的模型做预测
preds=model.predict_classes(X_test)

sub_set_1=X_test[Y_test ==1]
sub_set_2=X_test[preds ==1]
inter_set=[]

for item in sub_set_1.tolist():
    if item in sub_set_2.tolist():
        inter_set.append(item)

coverage=len(inter_set)/len(sub_set_1)
precentage_bad=sum(preds)/len(preds)
#输出预测结果
print("\nCoverage of default sample: ", coverage)
print("\nPrecentage of default preds: ", precentage_bad)

loss=train_process.history['loss']
accuracy=train_process.history['acc']
#画出模型训练曲线,如图7.14所示
epochs=range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.title('Training accuracy')
```

```
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
plt.title('Training loss')
plt.legend()
plt.show()
```

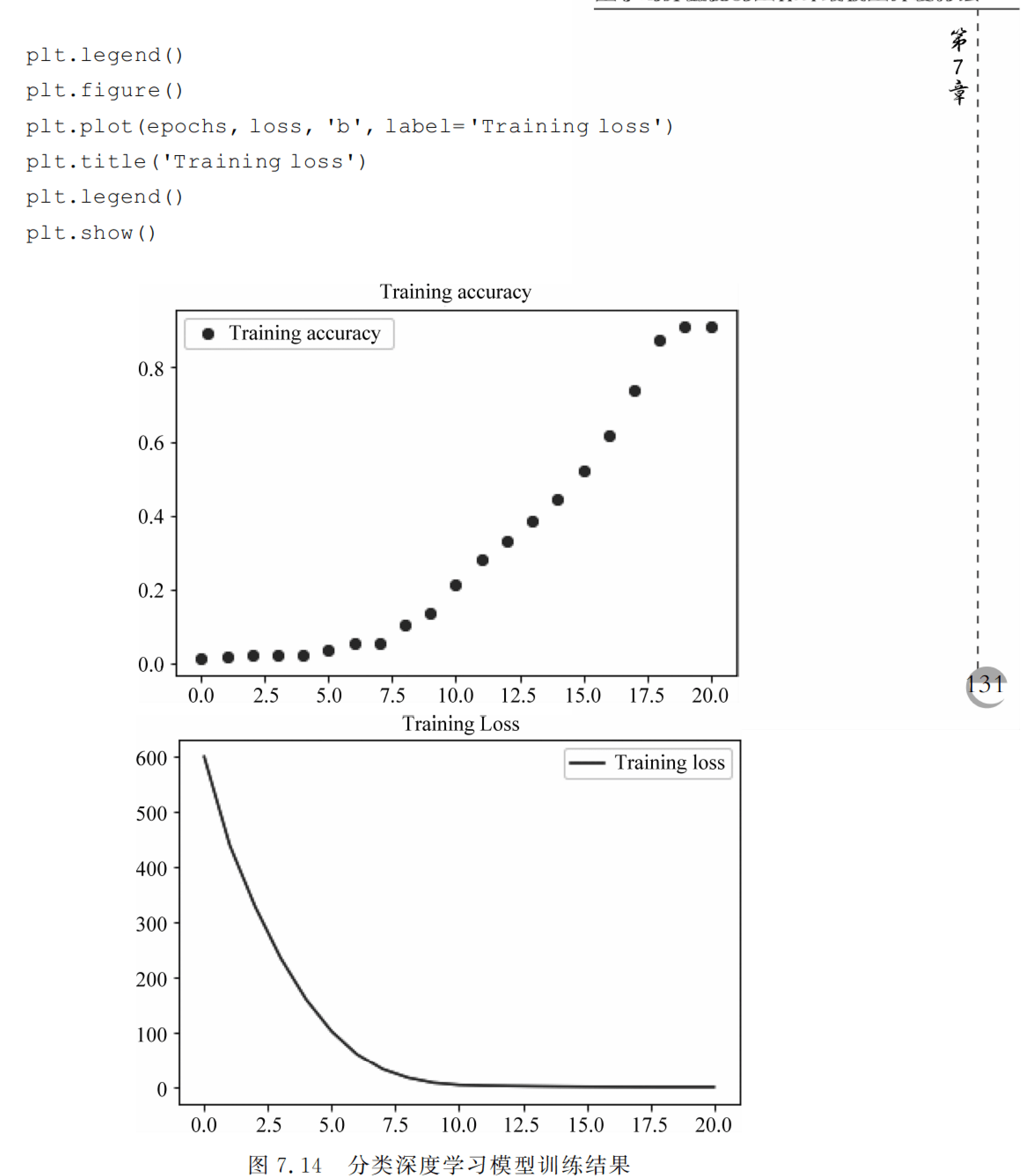


图 7.14　分类深度学习模型训练结果

由图 7.14 所示的训练结果曲线可知，随着训练次数的不断增加，模型的准确率逐渐提升、损失率逐渐降低。

第 8 章 主体评级模型开发方法

本书第 7 章中讲述的基于场外真实数据的评级模型开发方法的前提是获取这些场外数据，并提取相关的入模指标。但是，场外数据的获取和加工需要较强的 IT 技能，这对非计算机专业出身的量化投资研究人员来说是一个巨大的挑战。本章重点讲述通过模型结构的调整和优化来缓释财务粉饰造成影响的方法，这对无法获取场外真实数据的量化投资研究人员来说，也是非常实用的方法。

8.1 基于财务数据的评分卡模型缓释财报粉饰的基本原理

通常情况下，我们开发信用债发行主体的评级模型时，重点考虑的是发债主体最近 1 年中各季度财务变化的情况，及其对发行人还款能力的影响。但是，由于企业财报粉饰现象的普遍存在，这种仅仅考虑最近 1 年财务数据的评分卡模型，时常会产生较大的误判情况。因此，本节详细讲述的是笔者多年研究的针对目前国内市场降低财务数据粉饰对企业信用评估影响的实用方法。

笔者研究发现，已经被证实存在财务粉饰公司的财务指标一般表现为波动性大，为了缓释财务粉饰带来的负面影响，在模型结构设计时我们一般需要输入最近 3 年的财务数据，以便于衡量被评估主体的财务波动性情况。模型输入部分的结构设计，如表 8.1 所示。

表 8.1　输入 3 年企业财务数据

类别	指标	2014 年财务数据	2015 年财务数据	2016 年财务数据
盈利能力	净资产收益率(平均)	1.531 1	−9.602 9	−55.679 7
	营业利润/营业总收入	−8.430 3	−16.521 3	−70.627 7
收益质量	所得税/利润总额	13.106 94	11.095 616 67	14.903 52
现金流量	全部资产现金回收率	1.62 736 206	−0.679 165 481	2.038 559 54
资本结构	长期资本负债率	14.495 305 66	9.879 661 979	14.307 537 17

续表

类别	指标	2014 年财务数据	2015 年财务数据	2016 年财务数据
偿债能力	经营活动产生的现金流量净额/负债合计	0.027 9	−0.011 9	0.025 8
营运能力	应付财款周转天数	68.966 534 25	92.459 574 81	118.383 005 1
	总资产周转率	0.366 7	0.379 8	0.385 8
规模效应	筹资活动产生的现金流量净额	356 565 511.6	−197 191 351.7	−126 263 627.6
	投资活动产生的现金流量净额	66 501 524.58	−37 849 167.67	−28 225 730.87

模型的定量评分由三部分构成，分别是由最近 1 年财务数据得到的同行业评分、由最近 3 年财务数据得到的趋势评分和由最近 3 年财务数据得到的波动率评分。这样，我们强制入模指标的趋势和波动率进入模型，就可较大程度地降低仅仅采用财务数据本身时，由于财务粉饰对模型造成的影响。

模型定量的第一部分是同行业得分，如表 8.2 所示。该部分中每个指标的得分是通过查询表 8.3 所示的、通过统计同行业数据得到的评分表得到的。

表 8.2　基于最新 1 年财务数据评分

同行业	指标	得分
盈利能力	净资产收益率(平均)	0
	营业利润/营业总收入	0
收益质量	所得税/利润总额	60
现金流量	全部资产现金回收率	50
资本结构	长期资本负债率	60
偿债能力	经营活动产生的现金流量净额/负债合计	40
营运能力	应付账款周转天数	90
	总资产周转率	50
规模效应	筹资活动产生的现金流量净额	10
	投资活动产生的现金流量净额	10

表 8.3 所示的同行业得分表是通过全行业数据统计计算得到的，详细的计算方法在 8.2 节有详述，并附上了详细的 Python 代码。

表 8.3　最新 1 年财务数据评分表

同行业得分		10	20	30	40	50
盈利能力	净资产收益率(平均)	−9.433 127 273	0.468 672 727	1.176 354 545	1.608 872 727	2.576 872 727
	营业利润/营业总收入	−23.077 145 45	−12.280 209 09	−3.294 709 091	0.112 790 909	1.495 463 636
收益质量	所得税/利润总额	0.323 136 364	1.221 145 455	3.042 345 455	7.002 054 545	10.302 518 18
现金流量	全部资产现金回收率	−8.456 245 098	−2.117 532 123	−0.399 965 027	0.648 489 226	1.959 663 247
资本结构	长期资本负债率	52.364 539 83	40.420 454 91	30.857 196 2	25.687 970 35	21.415 392 56
偿债能力	经营活动产生的现金流量净额/负债合计	−0.168 527 273	−0.049 9	−0.010 363 636	0.013 536 364	0.043 463 636
营运能力	应付账款周转天数	12.533 617 27	18.548 346 47	28.493 848 35	35.431 404 7	44.052 348 93
	总资产周转率	0.075 554 545	0.170 727 273	0.213 518 182	0.279 245 455	0.359 854 545
规模效应	筹资活动产生的现金流量净额	−236 923 726.3	−82 027 886.46	−27 646 754.17	5 848 002.459	62 362 543.1
	投资活动产生的现金流量净额	23 193 276.31	−30 281 540.58	−55 950 209.33	−107 878 410.4	−175 543 100

同行业得分		60	70	80	90	100
盈利能力	净资产收益率(平均)	3.558 190 909	4.961 090 909	6.379 463 636	12.611 236 36	18.694 5
	营业利润/营业总收入	3.268 972 727	6.012 681 818	9.202 572 727	16.508 854 55	27.230 636 36
收益质量	所得税/利润总额	12.793 790 91	16.151 127 27	19.680 386 6	25.498 763 64	37.114 590 91
现金流量	全部资产现金回收率	3.380 194 011	4.744 680 995	7.530 803 378	10.813 541 85	14.935 225 51
资本结构	长期资本负债率	15.814 519 38	10.359 330 64	7.270 926 483	4.005 518 836	2.376 947 52
偿债能力	经营活动产生的现金流量净额/负债合计	0.0717 363 64	0.125 363 636	0.184 1	0.248 945 455	0.415 545 455
营运能力	应付账款周转天数	56.755 084 16	64.686 789 32	78.586 106 06	96.269 843 47	125.764 542 9
	总资产周转率	0.426 563 636	0.485 136 364	0.593 072 727	0.719 272 727	1.079 8
规模效应	筹资活动产生的现金流量净额	173 218 429.6	336 931 199	562 067 135.7	1 061 340 827	1 743 254 582
	投资活动产生的现金流量净额	−250 293 022.1	−422 700 168.1	−635 665 780.2	−952 449 458.9	−1 791 786 833

表 8.4 是计算每个入模指标最近 3 年趋势变化的得分表。在计算趋势得分时，我们需要首先计算每个指标最近 3 年的变化趋势，计算公式为

$$\text{Trend}(p) = 100 \times \frac{p_{2016} - p_{2014}}{\text{ABS}(p_{2014})} \tag{8.1}$$

表 8.4　计算最近 3 年财务数据趋势的变化

趋势	指标	绝对增长率	得分
盈利能力	净资产收益率(平均)	−3 736.581 543	0
	营业利润/营业总收入	−737.783 946	0
收益质量	所得税/利润总额	13.707 089 53	50
现金流量	全部资产现金回收率	25.267 731 74	50
资本结构	长期资本负债率	−1.295 374 425	70
偿债能力	经营活动产生的现金流量净额/负债合计	−7.526 881 72	40
营运能力	应付账款周转天数	71.652 826 09	80
	总资产周转率	5.208 617 398	80
规模效应	筹资活动产生的现金流量净额	−135.411 060 1	10
	投资活动产生的现金流量净额	−142.443 735	70

然后，再通过查询表 8.5 所示的趋势变化评分表得到的趋势变化的得分。表 8.5 所示的趋势变化评分表的计算方法，将会在 8.2 节中详述并提供 Python 源代码。

表 8.5　最近 3 年财务趋势变化评分表

趋势得分		10	20	30	40	50
盈利能力	净资产收益率(平均)	−187.489 597 6	−81.595 000 9	−63.763 921 5	−55.025 414 27	−39.042 941 22
	营业利润/营业总收入	−547.847 317 4	−158.957 181 8	−83.661 840 43	−5 014 721 872	−27.128 560 15
收益质量	所得税/利润总额	−90.699 124 31	−62.794 813 22	−39.533 116 1	−11.620 061 24	6.765 720 855
现金流量	全部资产现金回收率	−212.596 982 8	−114.629 193 3	−77.080 528 82	−39.332 978 95	−9.552 148 149
资本结构	长期资本负债率	487.528 673 4	145.687 104 5	79.225 782 28	44.139 364 11	22.269 118 43
偿债能力	经营活动产生的现金流量净额/负债合计	−246.961 273 5	−111.248 886	−80.524 357 07	−40.296 409 8	7 064 462 098
营运能力	应付账款周转天数	−37.354 925 05	−19.931 331 48	−10.891 746 68	−1.874 760 966	8.707 126 234
	总资产周转率	−47.518 500 22	−35.191 141 28	−29.407 078 66	−22.563 298 52	−16.553 694 61
规模效应	筹资活动产生的现金流量净额	−226.148 156	−135.335 176 9	−89.353 901 26	−49.733 902 97	−8.988 723 372
	投资活动产生的现金流量净额	115.604 348 7	84.633 623 66	51.152 919 02	21.182 798 32	−3.975 238 355

续表

趋势得分		60	70	80	90	100
盈利能力	净资产收益率(平均)	−22.585 030 89	−9.205 968 568	38.959 684 43	113.283 406 2	244.294 222 7
	营业利润/营业总收入	−6.869 091 73	12.689 599 06	40.037 203 2	83.602 265 71	226.774 811 2
收益质量	所得税/利润总额	23.302 909 71	42.669 859 92	78.733 873 04	173.923 784 2	473.351 752 5
现金流量	全部资产现金回收率	34.992 570 04	84.149 824 34	125.770 733 9	236.919 028 8	374.346 665 9
资本结构	长期资本负债率	10.599 515 28	1.019 113 322	−28.748 955 29	−44.629 484	−72.549 131 65
偿债能力	经营活动产生的现金流量净额/负债合计	36.829 810 89	95.805 191 98	128.271 178 4	230.705 090 5	471.922 261 7
营运能力	应付账款周转天数	24.049 640 06	40.036 382 66	57.829 865 71	104.934 827 5	200.097 452 4
	总资产周转率	−11.760 283 26	−7.400 458 528	2.160 873 173	13.662 980 71	25.596 692 25
规模效应	筹资活动产生的现金流量净额	64.531 184 38	101.976 280 5	163.844 643 3	418.954 379 2	1 561.822 505
	投资活动产生的现金流量净额	−44.153 023 15	−117.749 322 2	−205.691 161 3	−355.309 129 3	−842.446 732 4

表 8.6 是计算每个入模指标最近 3 年波动率变化的得分表。在计算波动率得分时,我们需要首先计算每个指标最近 3 年的波动率变化趋势,计算公式为

$$\text{Volatility}(p) = \frac{\text{STDEV}(p_{2014}, p_{2015}, p_{2016})}{\text{IF}(\text{ABS}(\text{AVERAGE}(p_{2014}, p_{2015}, p_{2016})) = 0, 0.0001, \text{ABS}(\text{AVERAGE}(p_{2014}, p_{2015}, p_{2016})))} \tag{8.2}$$

表 8.6　计算最近 3 年财务变化的波动率

波动率		均值的绝对值	标准差	标准差/(均值的绝对值)	得分
盈利能力	净资产收益率(平均)	21.250 5	30.331 812 48	1.427 345 826	20
	营业利润/营业总收入	31.859 766 67	33.816 868	1.061 428 615	30
收益质量	所得税/利润总额	13.035 358 89	1.904 960 588	0.146 137 947	90
现金流量	全部资产现金回收率	0.995 585 373	1.464 876 67	1.471 372 23	40
资本结构	长期资本负债率	12.894 168 27	2.612 326 623	0.202 597 528	70
偿债能力	经营活动产生的现金流量净额/负债合计	0.013 933 333	0.022 396 949	1.607 743 654 5	30
营运能力	应付账款周转天数	93.269 704 72	24.718 194 32	0.265 018 469	40
	总资产周转率	0.377 433 333	0.009 767 463	0.025 878 644	100

续表

波动率		均值的绝对值	标准差	标准差/（均值的绝对值）	得分
规模效应	筹资活动产生的现金流量净额	11 036 844.09	301 330 766	27.302 258 1	0
	投资活动产生的现金流量净额	142 208.68	57 669 937.91	405.530 365 1	0

然后，再通过查询表 8.7 所示的波动率变化评分表得到的波动率变化的得分。表 8.7 所示的波动率变化评分表的计算方法，将会在 8.2 节中详述并提供 Python 源代码。

表 8.7　最近 3 年财务波动率变化评分表

波动率得分		10	20	30	40	50
盈利能力	净资产收益率（平均）	3.500 327 802	2.132 755 642	1.300 423 408	1.002 503 416	0.652 357 352
	营业利润/营业总收入	3.963 563 018	1.695 557 306	1.334 306 329	1.033 031 602	0.848 984 898
收益质量	所得税/利润总额	1.421 992 643	1.121 580 394	0.891 053 212	0.758 944 878	0.571 698 211
现金流量	全部资产现金回收率	6.208 964 177	2.585 244 772	1.866 918 72	1.514 738 014	1.139 615 99
资本结构	长期资本负债率	0.982 586 097	0.729 466 288	0.553 335 983	0.458 011 586	0.351 352 114
偿债能力	经营活动产生的现金流量净额/负债合计	5.548 785 743	2.592 015 715	1.954 733 365	1.560 529 676	1.195 966 669
营运能力	应付账款周转天数	0.661 010 611	0.458 158 395	0.332 764 109	0.269 051 874	0.235 686 331
	总资产周转率	0.434 042 867	0.288 014 666	0.241 151 586	0.207 323 287	0.169 308 094
规模效应	筹资活动产生的现金流量净额	5.862 234 14	2.628 712 925	1.990 393 746	1.632 514 456	1.397 343 042
	投资活动产生的现金流量净额	2.874 594 386	1.665 365 067	1.256 265 957	0.984 173 449	0.816 170 386
波动率得分		**60**	**70**	**80**	**90**	**100**
盈利能力	净资产收益率（平均）	0.516 656 015	0.422 323 002	0.302 366 376	0.211 118 246	0.135 400 398
	营业利润/营业总收入	0.539 389 349	0.381 464 694	0.273 732 203	0.195 117 554	0.102 266 499
收益质量	所得税/利润总额	0.489 508 446	0.392 495 77	0.285 274 14	0.191 351 312	0.125 597 669
现金流量	全部资产现金回收率	0.925 106 4	0.743 076 002	0.538 717 765	0.383 842 772	0.265 938 922
资本结构	长期资本负债率	0.285 566 096	0.213 735 544	0.172 754 038	0.115 470 554	0.068 764 767
偿债能力	经营活动产生的现金流量净额/负债合计	0.933 842 898	0.750 139 551	0.585 366 667	0.403 684 433	0.280 349 759

续表

波动率得分		60	70	80	90	100
营运能力	应付账款周转天数	0.192 328 429	0.162 165 463	0.131 551 369	0.100 922 635	0.076 878 59
	总资产周转率	0.136 442 644	0.116 029 517	0.085 994 061	0.071 906 905	0.059 074 521
规模效应	筹资活动产生的现金流量净额	1.169 245 731	0.942 749 615	0.752 314 548	0.573 779 868	0.332 657 966
	投资活动产生的现金流量净额	0.714 524 857	0.603 010 223	0.471 573 231	0.335 588 94	0.211 096 665

这样，汇总上述三部分的得分，即为模型中定量部分的总得分，如表 8.8 所示。模型定量总得分中各指标权重的计算方法，将会在 8.2 节中详述并提供 Python 源代码。

表 8.8　模型定量部分评分汇总

定量得分			权重	得分	加权得分
同行业	盈利能力	净资产收益率(平均)	3.5%	0	0.0
		营业利润/营业总收入	3.3%	0	0.0
	收益质量	所得税/利润总额	4.2%	60	2.5
	现金流量	全部资产现金回收率	3.6%	50	1.8
	资本结构	长期资本负债率	3.5%	60	2.1
	偿债能力	经营活动产生的现金流量净额/负债合计	4.3%	40	1.7
	营运能力	应付账款周转天数	3.3%	90	3.0
		总资产周转率	4.1%	50	2.1
	规模效应	筹资活动产生的现金流量净额	1.7%	10	0.2
		投资活动产生的现金流量净额	1.3%	10	0.1
趋势	盈利能力	净资产收益率(平均)	3.0%	0	0.0
		营业利润/营业总收入	2.5%	0	0.0
	收益质量	所得税/利润总额	3.2%	50	1.6
	现金流量	全部资产现金回收率	2.5%	50	1.3
	资本结构	长期资本负债率	2.1%	70	1.5
	偿债能力	经营活动产生的现金流量净额/负债合计	2.9%	40	1.2
	营运能力	应付账款周转天数	2.6%	80	2.1
		总资产周转率	3.3%	80	2.6
	规模效应	筹资活动产生的现金流量净额	2.3%	10	0.2
		投资活动产生的现金流量净额	3.5%	70	2.5

续表

定 量 得 分			权重	得分	加权得分
波动率	盈利能力	净资产收益率(平均)	2.6%	20	0.5
		营业利润/营业总收入	3.5%	30	1.1
	收益质量	所得税/利润总额	4.9%	90	4.4
	现金流量	全部资产现金回收率	2.9%	40	1.2
	资本结构	长期资本负债率	4.3%	70	3.0
	偿债能力	经营活动产生的现金流量净额/负债合计	3.8%	30	1.1
	营运能力	应付账款周转天数	5.6%	40	2.2
		总资产周转率	4.2%	100	4.2
	规模效应	筹资活动产生的现金流量净额	3.3%	0	0.0
		投资活动产生的现金流量净额	4.2%	0	0.0

模型定性部分开发的基本原理是 AHP 法(AHP 法的详细介绍,请见笔者另外一本书《基于 R 语言的证券公司信用风险计量和管理》中第五章第一节的内容)。在定性部分的层次设计中,我们通常分为两层,第一层是准则层,包括该行业的产业属性和公司属性两部分;第二层是指标层,分别将准则层中的两部分细化到不同的定性指标。模型定性部分的结构设计,如表 8.9 所示。模型定性总得分中各指标权重的计算方法,将会在 8.2 节中详述并提供 Python 源代码。

表 8.9 模型定性部分评分汇总

机构定性评分								
	准则层	指标层	指标和得分					
		评估指标	符号	选项说明	选项	指标得分	指标最终权重	最终加权得分
机构定性评分 F	产业属性 A1	经济政策影响	P1	A. 良好的宏观经济环境,国家有政策支持,行业发展可持续增长	A	100	11.3%	11.3
				B. 稳定的宏观经济环境,国家有政策支持,行业发展稳定				
				C. 比较稳定的宏观经济环境,基本不受政策影响,行业发展有小幅波动				
				D. 较为波动的宏观经济环境,受政策影响较大,行业发展有较大幅波动				
				E. 较大波动的宏观经济环境,受政策影响很大,行业发展有剧烈波动				



续表

机构定性评分								
	准则层	指标层	指标和得分					
		评估指标	符号	选项说明	选项	指标得分	指标最终权重	最终加权得分
机构定性评分F	产业属性A1	经济周期性	P2	A. 受经济周期的影响很小 B. 受经济周期的影响较小 C. 受经济周期的影响一致 D. 受经济周期的影响较大 E. 受经济周期的影响很大	B	80	8.7%	7.0
	公司属性A2	公司性质	P3	A. 中央国有企业 B. 地方国有企业或公众企业 C. 中外合资企业或外商独资企业 D. 民营企业 E. 其他企业	B	80	8.0%	6.4
		对外担保余额占净资产的比例	P4	A. <=10% B. >10%及<=20% C. >20%及<=30% D. >30%及<=40% E. >40%	A	100	7.4%	7.4
		核心管理层变动	P5	A. 近3年没有发生股东变更 B. 近3年有发生股东变更	A	100	8.3%	8.3
		近1年来是否存在被执行和失信被执行	P6	A. 否 B. 是	B	0	12.0%	0.0
		近1年来是否存在被银行冻结存款或起诉欠款	P7	A. 否 B. 是	A	100	12.0%	12.0
		近1年来是否存在被股东起诉	P8	A. 否 B. 是	A	100	12.0%	12.0

续表

机构定性评分								
	准则层	指标层	指标和得分					
		评估指标	符号	选项说明	选项	指标得分	指标最终权重	最终加权得分
机构定性评分F	企业属性A2	近1年来是否存在被小贷(民间借贷)公司起诉	P9	A. 否	A	100	12.0%	12.0
				B. 是				
		招聘信息	P10	A. 近3年招聘职位数＞15	D	0	8.3%	0.0
				B. 5＜近3年招聘职位数＜＝15				
				C. 0＜近3年招聘职位数＜＝5				
				D. 近3年招聘职位数＝0				
					机构定性得分：			76.4

信用风险评级模型包括定量、定性和综合调整三部分，综合调整部分主要是为了缓释信用风险自身的滞后性所设计的。因为，模型开发所需的入模指标必须有稳定的数据源才行，而这些数据源的提供往往在很大程度上依赖于发债主体自身的披露。在目前国内资本市场信息披露机制不完善的情况下，企业经常选择性披露相关信息，对企业还款能力影响较大的信息，时常突发披露。

因此，为了缓释这种影响较大的突发事件，我们设计了模型的综合调整部分，如表8.10所示。综合调整部分各指标的调整幅度，需要根据各行业的不同属性做不同的调整。

表8.10　模型综合调整部分

定量总得分		44.07
定性总得分		76.35
总得分		47.06
初始内部等级		10
最近1年被诉且需要赔款的次数占所有同类案件的比重	无被诉且赔款记录	0
最近半年被执行次数占总数的比	无被执行记录	0
最近半年失信被执行次数占总数的比	无失信被执行记录	0
初始调整后内部等级		10
财报的质量	标准无保留意见	0

续表

财务披露的及时性	可获得最新季报	0
最近 3 个会计年度内，审计单位发生过变更	否	0
集团或母公司调整	请注明调整原因 （文本框，如有调整，需要注明原因）	0
突发重大事件调整	请注明调整原因 （文本框，如有调整，需要注明原因）	0
最终内部等级		10
最终评级		BB

综上，本节重点介绍的是缓释财务粉饰的模型架构设计方法。在实际的模型开发过程中，我们需要首先依据最新一年的财务数据和标记的违约样本做入模指标筛选，待确定入模指标后，需要将这些指标的趋势和波动率变化趋势指标强制进入模型，来开发评分卡模型。详细的模型开发步骤和方法，见 8.2 节。

8.2 升级版主体评级模型开发方法及 Python 源代码

本节在 8.1 节介绍缓释财务粉饰对企业信用评级模型开发影响的基础上，重点详细介绍该类升级版模型的开发方法，并给出了详细的 Python 源代码。运行本节的代码前，需要按照本书第 6 章中介绍的方法建设数据库并下载所有所需的数据之后，才能直接运行。本段代码可直接生成 8.1 节中介绍的评分卡模型的 Excel 版本，请读者认真研究。

```
#加载所需的 Python 包
import pymysql
import numpy as np
import pandas as pd
import openpyxl
import copy
import csv
import math
from openpyxl.utils.dataframe import dataframe_to_rows
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn import cross_validation, metrics
#配置数据库链接。需要特别说明的是，本段代码不能直接运行，需要首先按照本书
```

续表

财务披露的及时性	可获得最新季报	0
最近 3 个会计年度内，审计单位发生过变更	否	0
集团或母公司调整	请注明调整原因 （文本框，如有调整，需要注明原因）	0
突发重大事件调整	请注明调整原因 （文本框，如有调整，需要注明原因）	0
最终内部等级		10
最终评级		BB

综上，本节重点介绍的是缓释财务粉饰的模型架构设计方法。在实际的模型开发过程中，我们需要首先依据最新一年的财务数据和标记的违约样本做入模指标筛选，待确定入模指标后，需要将这些指标的趋势和波动率变化趋势指标强制进入模型，来开发评分卡模型。详细的模型开发步骤和方法，见 8.2 节。

8.2 升级版主体评级模型开发方法及 Python 源代码

本节在 8.1 节介绍缓释财务粉饰对企业信用评级模型开发影响的基础上，重点详细介绍该类升级版模型的开发方法，并给出了详细的 Python 源代码。运行本节的代码前，需要按照本书第 6 章中介绍的方法建设数据库并下载所有所需的数据之后，才能直接运行。本段代码可直接生成 8.1 节中介绍的评分卡模型的 Excel 版本，请读者认真研究。

```
#加载所需的 Python 包
import pymysql
import numpy as np
import pandas as pd
import openpyxl
import copy
import csv
import math
from openpyxl.utils.dataframe import dataframe_to_rows
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn import cross_validation, metrics
#配置数据库链接。需要特别说明的是，本段代码不能直接运行，需要首先按照本书
```

6.1 节中介绍的数据库配置并抓取相关数据后，本段代码才能直接运行。

```
config={
        'host':'localhost',
        'port':3306,
        'user':'test',
        'passwd':'admin',
        'db':'creditrisk',
        'charset':'gbk',
        'cursorclass':pymysql.cursors.DictCursor
        }
con=pymysql.connect(**config)

#读取 Excel 文件
def readwb(wbname,sheetname):
    wb=openpyxl.load_workbook(filename=wbname,read_only=True)
    if (sheetname==""):
        ws=wb.active
    else:
        ws=wb[sheetname]
    data=[]
    for row in ws.rows:
        list=[]
        for cell in row:
            aa=str(cell.value)
            if (aa==""):
                aa="1"
            list.append(aa)
        data.append(list)

    print (wbname +"-"+sheetname+"-已成功读取")
    return data
```

标记违约债券时，我们使用 Wind 信用债研究中的"负面事件报表""违约债券报表"和"违约兑付报表"中的债券发行主体。首先，在 Wind 首页窗口右下角输入"BCBA"，并按回车键，如图 8.1 所示。

在弹出的"信用债研究"窗口中，分别选择"负面事件报表""违约债券报表"和"违约兑付报表"，并将这三张表分别导出到 Excel 文件中，如图 8.2 所示。



图 8.1 Wind 信用债研究



图 8.2　导出违约和负面事件列表

```
#标记实质性违约样本
def mark_default(preprocessed_set):
    #读取图 8.2 中导出的三张报表,此处假设这三张报表的存放位置为 D:/Temp/风控模型
    default_list_1=readwb("D:/Temp/风控模型/负面事件报表.xlsx","Wind资讯")
    default_list_1=pd.DataFrame(default_list_1[1:(len(default_list_1)-2)],columns=default_list_1[0])
    default_list_2=readwb("D:/Temp/风控模型/违约兑付报表.xlsx","Wind资讯")
    default_list_2=pd.DataFrame(default_list_2[1:(len(default_list_2)-2)],columns=default_list_2[0])
    default_list_3=readwb("D:/Temp/风控模型/违约债券报表.xlsx","Wind资讯")
    default_list_3=pd.DataFrame(default_list_3[1:(len(default_list_3)-2)],columns=default_list_3[0])

    default_list_sub_1=pd.DataFrame({'COMP_NAME':default_list_1['发行人'], 'Date':default_list_1['发生日期']},columns=['COMP_NAME','Date'])
    default_list_sub_2=pd.DataFrame({'COMP_NAME':default_list_2['发行人全称'], 'Date':default_list_2['付息日']},columns=['COMP_NAME','Date'])
    default_list_sub_3=pd.DataFrame({'COMP_NAME':default_list_3['发行人'], 'Date':default_list_3['发生日期']},columns=['COMP_NAME',
```

```
'Date'])

    default_list=default_list_sub_1.append(default_list_sub_2).
append(default_list_sub_3).drop_duplicates()
    default_list=default_list.sort_values(['COMP_NAME','Date'],
ascending=True)

    distinct_list=pd.DataFrame()
    distinct_list=distinct_list.append(default_list.iloc[1,])
    #删除重复的公司
    for i in range(1,len(default_list)):
        if default_list.iloc[i,0] in list(distinct_list.COMP_NAME):
            continue
        else:
            distinct_list=distinct_list.append(default_list.iloc
[i,])

    distinct_list.Date=distinct_list.Date.astype(str)
    for i in range(len(distinct_list)):
        distinct_list.Date.iloc[i]=distinct_list.Date.iloc[i][0:4]
    distinct_list.Date=distinct_list.Date.astype(int)

    #标记违约债券的发行主体为1,不违约为0
    data_set=copy.copy(preprocessed_set)
    data_set['isDefault']=0
    data_set.rptDate=data_set.rptDate.astype(str)

    for i in range(len(data_set)):
        if data_set.COMP_NAME.iloc[i] in list(distinct_list.COMP_
NAME) and int(data_set.rptDate.iloc[i][0:4]) >=distinct_list
[distinct_list['COMP_NAME']==data_set.COMP_NAME.iloc[i]].Date.
iloc[0]-1:
            data_set.isDefault.iloc[i]=1

    return data_set
```

在国内资本市场,目前已经违约的信用债发行主体的数量是非常少的,占所有债券发行主体的不足1%。这样就出现了严重的样本不均衡问题,当我们直接使用这样的样本总体进行模型开发、统计检验时,很多算法不能直接给出结果。本书第2章介绍的解决样本不均衡问题的算法又不能直接应用于信用债发

行主体的财务数据，因为由第5、第6两章的统计数据和检验结果可见，财务数据无法真实有效地反映企业的真实还款能力和财务水平。

例如，很多已经违约的公司，其现金流和还款能力均表现“良好”。如果直接对已经违约样本的财务数据进行“过采样”，就会模拟出很多财务状况良好但已经违约的样本。就笔者实际的统计检验结果来看，这种做法更使得财务数据对违约状况影响不显著，基本无法将任何财务指标进入模型。

此处，我们的解决方案是采用自定义技术性违约样本的方法来解决样本严重不均衡问题。笔者在使用财务数据开发评分卡模型时，通常采用“现金到期债务比”“现金流量利息保障倍数”和“经营活动产生的现金流量净额/流动负债”三个指标来自定义技术性违约。我们首先在每个行业中均选择这三个指标排名最后20%的公司，然后取这些公司的交集，作为该行业的技术性违约样本。

```
    #标记技术性违约
def mark_technical_default(marked_set,percentage):
    #marked_set=copy.copy(data_set)
    #percentage=0.2
    rptDate_list=['20161231','20151231','20141231','20131231','20121231']
    for rptDate in rptDate_list:
        #rptDate='20161231'
        data_set=marked_set[marked_set['rptDate']==rptDate]
        n=len(data_set)
        start_num=n-int(percentage * n)
        sub_set=data_set.sort_values('OCFTOQUICKDEBT',ascending=False)
        sub_set=sub_set.iloc[start_num:n,]
        code_1=sub_set.CODE
        sub_set=data_set.sort_values('OCFTOINTEREST',ascending=False)
        sub_set=sub_set.iloc[start_num:n,]
        code_2=sub_set.CODE
        sub_set=data_set.sort_values('OCFTOSHORTDEBT',ascending=False)
        sub_set=sub_set.iloc[start_num:n,]
        code_3=sub_set.CODE

        #对这三个指标取交集
        codes=[]
        for i in range(len(data_set)):
```

```
            if data_set.CODE.iloc[i] in list(code_1) and data_set.
CODE.iloc[i] in list(code_2) and data_set.CODE.iloc[i] in list(code_3):
                codes.append(data_set.CODE.iloc[i])

        #标记技术性违约
        for i in range(len(data_set)):
            if data_set.CODE.iloc[i] in codes:
                marked_set.loc[(marked_set['rptDate'] ==rptDate) &
(marked_set['CODE'] ==data_set.CODE.iloc[i]),'isDefault']=1

    return marked_set

#计算误差平方和
def rssError(yArr, yHatArr):
    return ((yArr -yHatArr) ** 2).sum()
#标准化
def regularize(xMat):  #regularize by columns
    inMat=xMat.copy()
    inMeans=np.mean(inMat, 0)  #calc mean then subtract it off
    inVar=np.var(inMat, 0)  #calc variance of Xi then divide by it
    inMat=(inMat -inMeans) / inVar
    return inMat
#逐步回归筛选入模指标
def stepWise(xArr, yArr, step=0.01, numIt=5000):
    #xArr=copy.copy(index)
    #yArr=copy.copy(isDefault)
    xMat=np.mat(xArr)
    xMat=regularize(xMat)
    yMat=np.mat(yArr).T
    yMean=np.mean(yMat)
    yMat=yMat -yMean
    N, n=np.shape(xMat)
    returnMat=np.zeros((numIt, n))
    ws=np.zeros((n, 1))
    wsTest=ws.copy()
    weMax=ws.copy()
    for ii in range(numIt):
        #print(ws.T)
        lowestErr=np.inf
        for jj in range(n):
```

```
            for sign in [-1, 1]:
                wsTest=ws.copy()
                wsTest[jj] +=step * sign
                yTest=xMat * wsTest
                rssE=rssError(yMat.A, yTest.A)
                if rssE <lowestErr:
                    lowestErr=rssE
                    wsMax=wsTest
        ws=wsMax.copy()
        returnMat[ii, :]=ws.T
    return returnMat


##数据预处理及入模指标筛选
def index_select(industry,percentage):
    #industry="采矿业"
    #percentage=0.2
    #从数据库读取,指定行业的所有数据
    try:
        with con.cursor() as cursor:
             query="SELECT * FROM creditrisk.all_ratio_data WHERE
industry_L1='" +industry +"';"
            cursor.execute(query)
            result=cursor.fetchall()
    finally:
        con.close
    #删除定性相关指标
    ratio_data=pd.DataFrame(result)
    del_list=['CITY','CITYINVESTMENTBONDGEO','MUNICIPALBOND',
              'FOUNDDATE','HOLDER_PCT','HOLDER_SHARECATEGORY',
              'LISTINGORNOT','NATURE','PROVINCE','REGCAPITAL',
              'SHAREHOLDERNATURE','industry_L1','industry_L2',
              'industry_L1_wind','industry_L2_wind','industry_
              L3_wind','industry_L4_wind']
    for col_name in del_list:
        del ratio_data[col_name]

    #df.head
    name_list=list(ratio_data)
    protect_list=['OCFTOQUICKDEBT','OCFTOINTEREST','OCFTOSHORTDEBT']
```

```
#去除空缺值过多的指标,如果缺失值大于1/3,则删除该指标
n=ratio_data.columns.size
m=ratio_data.iloc[:,0].size
adjust=0
for i in range(n):
    #i=1
    data_list=ratio_data.iloc[:,i -adjust]
    na_count=0
    if name_list[i] in protect_list:
        continue
    for j in range(m):
        #j=1
        if data_list[j] !=data_list[j]:
            na_count +=1
    if (na_count/m)>(1/3):
        del ratio_data[name_list[i]]
        adjust +=1
```



```
#去除空缺值过多的样本,如果缺失值大于1/3,则删除该样本
n=ratio_data.columns.size
m=ratio_data.iloc[:,0].size
adjust=0
for i in range(m):
    #i=1
    data_list=ratio_data.iloc[i -adjust,:]
    na_count=0
    for j in range(n):
        #j=1
        if data_list[j] !=data_list[j]:
            na_count +=1
    if (na_count/n)>(1/3):
        ratio_data=ratio_data.drop(i)
        adjust +=1
```

缺失值的填充方法有很多,最简单的是用行业均值、中位数、众数等去填充。还有通过其他不含缺失值的指标,建立如线性回归、随机森林回归等的预测模型来填充缺失值的方法。这些方法在其他书中较常见,此处就不一一介绍了。

本书采用的缺失值填充方法,是笔者多年研究中国市场得出的深刻体会总结。笔者发现,中国市场同质化趋势明显,这主要表现为同一类型的公司,很多

指标尤其是比率型的指标趋势变化同质化。因此，此处我们采取的缺失值填充策略是根据企业的营业收入，分为四分位数。即根据企业的营业收入，分为大型公司、中大型公司、中型公司、小型公司，如果一个存在缺失值的样本属于大型公司，则用大型公司中所有非缺失值样本的平均数来填充该缺失指标，以此类推。

```
#填充缺失值
try:
    with con.cursor() as cursor:
        query="SELECT COMP_NAME,TOT_OPER_REV AS oper_rev,rptDate FROM creditrisk.all_ratio_data;"
        cursor.execute(query)
        result=cursor.fetchall()
finally:
    con.close
#读取企业营业收入
oper_rev=pd.DataFrame(result)
oper_rev=oper_rev.drop_duplicates()
comp_list=pd.DataFrame()
comp_list['CODE']=ratio_data.CODE
comp_list['COMP_NAME']=ratio_data.COMP_NAME
comp_list=comp_list.drop_duplicates()
oper_rev_list=pd.merge(comp_list,oper_rev,how='left',on='COMP_NAME')
rptDate_list=['20161231','20151231','20141231','20131231','20121231']

complete_set=pd.DataFrame()
count_nas=[0 for i in range(0,ratio_data.columns.size)]

for rptdate in rptDate_list:
    #rptdate='20161231'
    data_set=oper_rev_list[oper_rev_list['rptDate']==str(rptdate)]
    data_set=data_set.fillna(-1)
    data_set=data_set[data_set['oper_rev'] != (-1)]
    #用分位数,按照公司的营业收入将公司分为大型、中大型、中型和小型公司
    boundarys=[data_set['oper_rev'].quantile(0),data_set['oper_rev'].quantile(0.25), data_set['oper_rev'].quantile(0.5),data_set['oper_rev'].quantile(0.75), data_set['oper_rev'].quantile(1)]
    #获取不同类型公司的边界
    part_1_list=data_set[data_set['oper_rev'] <=boundarys[1]]
    del part_1_list['oper_rev']
    part_2_list=data_set[data_set['oper_rev']>boundarys[1]]
```

```
part_2_list=part_2_list[part_2_list['oper_rev'] <=boundarys[2]]
del part_2_list['oper_rev']
part_3_list=data_set[data_set['oper_rev']>boundarys[2]]
part_3_list=part_3_list[part_3_list['oper_rev'] <=boundarys[3]]
del part_3_list['oper_rev']
part_4_list=data_set[data_set['oper_rev']>boundarys[3]]
del part_4_list['oper_rev']
#合并整理不同类型公司的数据集
part_1=pd.merge(part_1_list,ratio_data,how='inner',
                left_on=['CODE','rptDate','COMP_NAME'],
                right_on=['CODE','rptdate','COMP_NAME'])
del part_1['rptdate']
part_2=pd.merge(part_2_list,ratio_data,how='inner',
                left_on=['CODE','rptDate','COMP_NAME'],
                right_on=['CODE','rptdate','COMP_NAME'])
del part_2['rptdate']
part_3=pd.merge(part_3_list,ratio_data,how='inner',
                left_on=['CODE','rptDate','COMP_NAME'],
                right_on=['CODE','rptdate','COMP_NAME'])
del part_3['rptdate']
part_4=pd.merge(part_4_list,ratio_data,how='inner',
                left_on=['CODE','rptDate','COMP_NAME'],
                right_on=['CODE','rptdate','COMP_NAME'])
del part_4['rptdate']

name_list_part=list(part_1)
n=part_1.columns.size

part_1_means=part_1.iloc[:,3:n].describe().mean()#计算大型公司该指标的平均数
part_2_means=part_2.iloc[:,3:n].describe().mean()#计算中大型公司该指标的平均数
part_3_means=part_3.iloc[:,3:n].describe().mean()#计算中型公司该指标的平均数
part_4_means=part_4.iloc[:,3:n].describe().mean()#计算小型公司该指标的平均数
#填充大型公司的缺失值
for i in range(3,n):
    #i=4
    data_list=part_1.iloc[:,i]
    na_count=0
    for j in range(len(part_1)):
```

```
        #j=1
        if data_list[j] !=data_list[j]:
            na_count +=1
            part_1.iloc[j,i]=part_1_means[i-3]
    count_nas[i] +=na_count
#填充中大型公司的缺失值
for i in range(3,n):
    #i=4
    data_list=part_2.iloc[:,i]
    na_count=0
    for j in range(len(part_2)):
        #j=1
        if data_list[j] !=data_list[j]:
            na_count +=1
            part_2.iloc[j,i]=part_2_means[i-3]
    count_nas[i] +=na_count
#填充中型公司的缺失值
for i in range(3,n):
    #i=4
    data_list=part_3.iloc[:,i]
    na_count=0
    for j in range(len(part_3)):
        #j=1
        if data_list[j] !=data_list[j]:
            na_count +=1
            part_3.iloc[j,i]=part_3_means[i-3]
    count_nas[i] +=na_count
#填充小型公司的缺失值
for i in range(3,n):
    #i=4
    data_list=part_4.iloc[:,i]
    na_count=0
    for j in range(0,len(part_4)):
        #j=1
        if data_list[j] !=data_list[j]:
            na_count +=1
            part_4.iloc[j,i]=part_4_means[i-3]
    count_nas[i] +=na_count

data_set=part_1.append(part_2).append(part_3).append(part_4)
complete_set=complete_set.append(data_set)
```

```
count_nas=pd.Series(count_nas, index=list(complete_set))

##违约标记##
complete_set=mark_technical_default(mark_default(complete_set),
percentage)

#complete_set[complete_set['isDefault']==1]
ratio_type_list=[['ROE_AVG','ROE_BASIC','ROE_DILUTED','ROE_
DEDUCTED','ROE_EXBASIC','ROE_EXDILUTED','ROE_ADD','ROA2','ROA',
'ROIC','ROE_YEARLY','ROA2_YEARLY','ROA_YEARLY','NETPROFITMARGIN',
'GROSSPROFITMARGIN', ' COGSTOSALES ', ' NPTOCOSTEXPENSE ',
'EXPENSETOSALES',' OPTOEBT ',' PROFITTOGR ',' OPTOGR ',' EBITTOGR ',
'GCTOGR',' OPERATEEXPENSETOGR ',' ADMINEXPENSETOGR ',' FINAEXPENSETOGR ',
'IMPAIRTOGR','IMPAIRTOOP','EBITDATOSALES'],
[' OPERATEINCOMETOEBT ',' INVESTINCOMETOEBT ',' NONOPERATEPROFITTOEBT ',
'TAXTOEBT','DEDUCTEDPROFITTOPROFIT'],
[' SALESCASHINTOOR ', ' OCFTOOR ', ' OCFTOOPERATEINCOME ',
'CAPITALIZEDTODA',' OCFTOCF ',' ICFTOCF ',' FCFTOCF ',' OCFTOSALES ',
'OCFTOINVESTSTOCKDIVIDEND','OCFTOOP','OCFTOASSETS','OCFTODIVIDEND'],
[' DEBTTOASSETS ',' DEDUCTEDDEBTTOASSETS ',' LONGDEBTTOLONGCAPTIAL ',
'LONGCAPITALTOINVESTMENT', ' ASSETSTOEQUITY ', ' CATOASSETS ',
'CURRENTDEBTTOEQUITY', ' NCATOASSETS ', ' LONGDEBTTOEQUITY ',
'TANGIBLEASSETTSTOASSETS',' EQUITYTOTOTALCAPITAL ',' INTDEBTTOTALCAP ',
'CURRENTDEBTTODEBT','LONGDEBTTODEBT','NCATOEQUITY'],
[' CURRENT ', ' QUICK ', ' CASHRATIO ', ' CASHTOCURRENTDEBT ',
'OCFTOQUICKDEBT',' OCFTOINTEREST ',' DEBTTOEQUITY ',' EQUITYTODEBT ',
'EQUITYTOINTERESTDEBT',' TANGIBLEASSETTODEBT ',' TANGASSETTOINTDEBT ',
'TANGIBLEASSETTONETDEBT',' DEBTTOTANGIBLEEQUITY ',' EBITDATODEBT ',
'OCFTODEBT','OCFTOINTERESTDEBT','OCFTOSHORTDEBT','OCFTOLONGDEBT',
' OCFTONETDEBT ', ' OCFICFTOCURRENTDEBT ', ' OCFICFTODEBT ',
'EBITTOINTEREST', ' LONGDEBTTOWORKINGCAPITAL ', ' LONGDEBTTODEBT ',
'NETDEBTTOEV', ' INTERESTDEBTTOEV ', ' EBITDATOINTERESTDEBT ',
'EBITDATOINTEREST','TLTOEBITDA','CASHTOSTDEBT'],
['TURNDAYS',' INVTURNDAYS ',' ARTURNDAYS ',' APTURNDAYS ',' NETTURNDAYS ',
'INVTURN',' ARTURN ',' CATURN ',' OPERATECAPTIALTURN ',' FATURN ',' NON_
CURRENTASSETSTURN','ASSETSTURN1','APTURN'],
['TOT_ASSETS','TOT_LIAB','TOT_EQUITY','TOT_OPER_REV','OPPROFIT',
'NET_PROFIT_IS','NET_CASH_FLOWS_OPER_ACT','NET_CASH_FLOWS_INV_ACT',
'NET_CASH_FLOWS_FNC_ACT']]

final_name_list=['CODE','rptDate','COMP_NAME']
selected_name_list=[]
```

```
isDefault=complete_set.isDefault
#指标筛选和预处理
for x in range(7):
    #x=0
    col_list=[]
    for j in ratio_type_list[x]:
        if j in list(complete_set):
            col_list.append(j)

    index=complete_set.loc[:,col_list]
    r_mat=np.corrcoef(index,rowvar=0)
    index_names=list(index)
    drop_list=[]

    #首先剔除相关性较强(>0.8)的指标
    for i in range(0,len(r_mat)):
        if index_names[i] in drop_list:
            continue
        else:
            for j in range(0,len(r_mat)):
                if index_names[j] in drop_list:
                    continue
                else:
                     if i !=j and abs(r_mat[i,j]) >=0.8 and count_nas
[index_names[j]] >=count_nas[index_names[i]]:
                        drop_list.append(index_names[j])
                    else:
                        continue

    if len(drop_list)>0:
        for i in range(len(drop_list)):
            del index[drop_list[i]]

    if len(list(index)) ==1:
        selected_name_list.append(list(index)[0])
        continue

    #指标筛选,方法一:随机森林
    #No.1 : Random forest
    clf=RandomForestClassifier(oob_score=True, random_state=10,
class_weight=None)
    clf.fit(index,isDefault)
```

```
    #list(index)
    #list(clf.feature_importances_)
    rf_importance=pd.DataFrame({'name' : list(index), 'importance' :
list(abs(clf.feature_importances_))})
    rf_importance=rf_importance.sort_values('importance',ascending
=False)
    selected_name_list.append(rf_importance.iloc[0,1])
    #指标筛选,方法二:逐步回归
    #No.2 : Front stepwise Regression,前向逐步回归算法,筛选入模指标
    wMat=stepWise(index,isDefault)
    sw_importance=pd.DataFrame({'name' : list(index), 'importance' :
list(abs(wMat[4999]))})
    sw_importance=sw_importance.sort_values('importance',ascending
=False)
    selected_name_list.append(sw_importance.iloc[0,1])

selected_name_list=list(set(selected_name_list))
selected_name_list_2=[]
for i in range(7):
    for j in range(len(selected_name_list)):
        if selected_name_list[j] in ratio_type_list[i]:
            selected_name_list_2.append(selected_name_list[j])

#二次去共线性
preselected_set=complete_set.loc[:,selected_name_list_2]
r_mat=np.corrcoef(preselected_set,rowvar=0)
index_names=list(preselected_set)
drop_list=[]

for i in range(0,len(r_mat)):
    if index_names[i] in drop_list:
        continue
    else:
        for j in range(0,len(r_mat)):
            if index_names[j] in drop_list:
                continue
            else:
                if i !=j and abs(r_mat[i,j]) >=0.8 and count_nas[index_
names[j]] >=count_nas[index_names[i]]:
                    drop_list.append(index_names[j])
                else:
                    continue
```

```
if len(drop_list)>0:
    for i in range(len(drop_list)):
        del preselected_set[drop_list[i]]

selected_name_list=list(preselected_set)

for i in range(len(selected_name_list)):
    final_name_list.append(selected_name_list[i])

final_name_list.append('isDefault')
selected_set=complete_set.loc[:,final_name_list]

return selected_set

#返回选中的入模指标
def list_select(input_set,column_name,select_list):
    output_set=pd.DataFrame(columns=list(input_set))
    for i in range(len(input_set)):
        if input_set[column_name].iloc[i] in select_list:
            output_set=output_set.append(input_set.iloc[i,:])
    return output_set

#计算各入模指标的同行业、趋势、波动率评分表
def calculate_factors(selected_set,year_end):
    #year_end=2016
    year_mid=year_end -1
    year_start=year_end -2
    rptDate_start=str(year_start) +'1231'
    rptDate_mid=str(year_mid) +'1231'
    rptDate_end=str(year_end) +'1231'

    data_set_start=selected_set[selected_set['rptDate'] ==rptDate_
start]
    data_set_mid=selected_set[selected_set['rptDate'] ==rptDate_
mid]
    data_set_end=selected_set[selected_set['rptDate'] ==rptDate_
end]

    code_1=list(data_set_start.CODE)
    code_2=list(data_set_mid.CODE)
    code_3=list(data_set_end.CODE)
```

```
#取交集
codes=[]
for i in range(len(selected_set.CODE)):
    if list(selected_set.CODE)[i] in code_1 and list(selected_
set.CODE)[i] in code_2 and list(selected_set.CODE)[i] in code_3:
        codes.append(list(selected_set.CODE)[i])
#去重
codes=list(set(codes))

#截选
data_set_start=list_select(data_set_start,'CODE',codes)
data_set_mid=list_select(data_set_mid,'CODE',codes)
data_set_end=list_select(data_set_end,'CODE',codes)

#排序
data_set_start=data_set_start.sort_values('CODE',ascending=
True)
data_set_mid=data_set_mid.sort_values('CODE',ascending=True)
data_set_end=data_set_end.sort_values('CODE',ascending=True)

output_set=data_set_end.loc[:,['CODE','COMP_NAME']]
isDefault=data_set_end['isDefault']

del_list=['CODE','rptDate','COMP_NAME','isDefault']
for col_name in del_list:
    del data_set_start[col_name]
    del data_set_mid[col_name]
    del data_set_end[col_name]

#结束年
col_list=list(data_set_end)
for col_name in col_list:
    output_set[col_name]=data_set_end[col_name]

#计算趋势(Trend)
for col_name in col_list:
    trend_name=col_name +'_T'
    temp=[]

    for i in range(len(data_set_end)):
        temp.append(100 * (data_set_end[col_name].iloc[i] -data-
```

```
set_start[col_name].iloc[i])/abs(data_set_start[col_name].iloc[i]))

        output_set[trend_name]=temp

    #计算波动性(Volitility)
    for col_name in col_list:
        volitility_name=col_name +'_V'
        temp=[]

        for i in range(len(data_set_end)):
            Sdv=np.std([data_set_start[col_name].iloc[i],data_set_
mid[col_name].iloc[i],data_set_end[col_name].iloc[i]])
            absMean=abs(np.mean([data_set_start[col_name].iloc[i],
data_set_mid[col_name].iloc[i],data_set_end[col_name].iloc[i]]))

            if absMean ==0:
                absMean=0.0001

            temp.append(Sdv/absMean)

        output_set[volitility_name]=temp

    output_set['isDefault']=isDefault

    return output_set

#取分位数
def quantile(data_list,seq_num=4):
    quantile_list=[]
    for i in range(seq_num+1):
        percentage=(i/seq_num) * 100
        quantile_list.append(np.percentile(data_list,percentage))
    return quantile_list

#把数据写入Excel文件
def write_dataframe_to_excel(sheet,df,start_col=1,start_row=1,
index=False,columns=True):
    #sheet=ws1
    column_list=['','A','B','C','D','E','F','G','H','I','J','K',
'L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
    writing_row=0
    for r in dataframe_to_rows(df, index=index, header=columns):
```

```
        for j in range(len(r)):
            cell_index=column_list[j +start_col] +str(start_row +
writing_row)
            sheet[cell_index].value=r[j]
        writing_row +=1


#读取 csv 文件
def read_csv(input_dir):
    csvFile=open(input_dir, "r")
    reader=csv.reader(csvFile)  #返回的是迭代类型
    data=[]
    for item in reader:
        #print(item)
        data.append(item)

    csvFile.close()
    return data
#计算几何平均
def geometric_mean(input_list):
    n=len(input_list)
    a=1
    for i in range(n):
        a=a * float(input_list[i])
    return pow(a,1/n)

#列表乘法
def multiply_list(input_list,y):
    def f(x):
        return x * y
    return list(map(f,input_list))

#列表除法
def devide_list(input_list,y):
    def f(x):
        return x / y
    return list(map(f,input_list))

#使用 AHP 法计算模型定性部分指标的权重
def get_qualitative_weigth(industry):
    #####目标层---准则层,权重的确定#####
    input_dir=" D:/Temp/风控模型/AHP-" +industry +"-重要性判断矩阵
```

```
1.csv"
    input_set=read_csv(input_dir)#读取目标层---准则层相对重要性矩阵
    #input_set=pd.DataFrame(input_set,columns=["A1","A2"],index=
["A1","A2"])
    gi_a1=geometric_mean(input_set[0])
    gi_a2=geometric_mean(input_set[1])
    total_gi_a1_a2=gi_a1 +gi_a2
    wi_a1=gi_a1 / total_gi_a1_a2
    wi_a2=gi_a2 / total_gi_a1_a2

    #####准则层 A1---指标层 P,权重的确定#####
    input_dir=" D:/Temp/风控模型/AHP-" +industry +"-重要性判断矩阵
2.csv"
    input_set=read_csv(input_dir) #读取准则层---指标层相对重要性矩阵
    a1p_p1=geometric_mean(input_set[0])
    a1p_p2=geometric_mean(input_set[1])
    total_gi_a1p=a1p_p1 +a1p_p2
    wi_a1p_p1=a1p_p1 / total_gi_a1p
    wi_a1p_p2=a1p_p2 / total_gi_a1p

    #####准则层 A2---指标层 P,权重的确定#####
    input_dir=" D:/Temp/风控模型/AHP-" +industry +"-重要性判断矩阵
3.csv"
    input_set=read_csv(input_dir) #读取准则层---指标层相对重要性矩阵
    a2p_p3=geometric_mean(input_set[0])
    a2p_p4=geometric_mean(input_set[1])
    a2p_p5=geometric_mean(input_set[2])
    a2p_p6=geometric_mean(input_set[3])
    a2p_p7=geometric_mean(input_set[4])
    a2p_p8=geometric_mean(input_set[5])
    a2p_p9=geometric_mean(input_set[6])
    a2p_p10=geometric_mean(input_set[7])
    a2p_p11=geometric_mean(input_set[8])
    total_gi_a2p=a2p_p3+a2p_p4+a2p_p5+a2p_p6+a2p_p7+a2p_p8+a2p_
p9+a2p_p10+a2p_p11
    wi_a2p_p3=a2p_p3 / total_gi_a2p
    wi_a2p_p4=a2p_p4 / total_gi_a2p
    wi_a2p_p5=a2p_p5 / total_gi_a2p
    wi_a2p_p6=a2p_p6 / total_gi_a2p
    wi_a2p_p7=a2p_p7 / total_gi_a2p
    wi_a2p_p8=a2p_p8 / total_gi_a2p
    wi_a2p_p9=a2p_p9 / total_gi_a2p
```

```
test_data=test_data.T
test_data.columns=["2014年财务数据","2015年财务数据","2016年财务数据"]
factor_title=pd.DataFrame({'type':type_list,'factor_names':factor_names},columns=['type','factor_names'])

row_to_write=1

wb=openpyxl.Workbook()
ws1=wb.active
ws1.title='定量评分'

write_dataframe_to_excel(ws1,factor_title,start_col=1,start_row=2,index=False,columns=False)
write_dataframe_to_excel(ws1,test_data,start_col=3,start_row=1,index=False,columns=True)
row_to_write +=len(factor_title) +2

#得分模块#
ws1["A% d" % row_to_write].value='得分'
row_to_write +=2

#同行业
ws1["A% d" % row_to_write].value='同行业'
ws1["F% d" % row_to_write].value='得分'
row_to_write +=1

write_dataframe_to_excel(ws1,factor_title,start_col=1,start_row=row_to_write,index=False,columns=False)
for i in range(len(factor_title)):
    rw=row_to_write +i
    formula="=IF(同行业!$ C"+str(i+3)+"<同行业!$ L"+str(i+3)+",IF(E"+str(i+3)+"<同行业!$ C"+str(i+3)+",0,IF(E"+str(i+3)+">同行业!$ L"+str(i+3)+",100,OFFSET(同行业!$ B$ 2,0,MATCH(E"+str(i+3)+",同行业!$ C"+str(i+3)+":$ L"+str(i+3)+")))) "+",IF(E"+str(i+2)+">同行业!$ C"+str(i+3)+",0,IF(E"+str(i+2)+"<同行业!$ L"+str(i+3)+",100,OFFSET(同行业!$ B$ 2,0,MATCH(E"+str(i+2)+",同行业!$ C"+str(i+3)+":$ L"+str(i+3)+",-1))))"+")"
    ws1["F% d" % rw].value=formula

row_stamp_1=row_to_write
row_to_write +=len(factor_title) +1
```

```
    #趋势
    ws1["A% d" %  row_to_write].value='趋势'
    ws1["C% d" %  row_to_write].value='绝对增长率'
    ws1["F% d" %  row_to_write].value='得分'
    row_to_write +=1

    write_dataframe_to_excel(ws1,factor_title,start_col=1,start_
row=row_to_write,index=False,columns=False)
    for i in range(len(factor_title)):
        rw=row_to_write +i
        formula="=100 * (E"+str(i+2)+"-C"+str(i+2)+")/ABS(C"+str(i
+2)+")"
        ws1["C% d" %  rw].value=formula
    for i in range(len(factor_title)):
        rw=row_to_write +i
        formula="=IF(趋势!$ C"+str(i+3)+"<趋势!$ L"+str(i+3)+",IF
(C"+str(rw)+"<趋势!$ C"+str(i+3)+",0,IF(C"+str(rw)+">趋势!$ L"+
str(i+3)+",100,OFFSET(趋势!$ B$ 2,0,MATCH(C"+str(rw)+",趋势!$ C"+
str(i+3)+":$ L"+str(i+3)+"))))"+",IF(C"+str(rw)+">趋势!$ C"+str(i
+3)+",0,IF(C"+str(rw)+"<趋势!$ L"+str(i+3)+",100,OFFSET(趋势!$ B
$ 2,0,MATCH(C"+str(rw)+",趋势!$ C"+str(i+3)+":$ L"+str(i+3)+",-
1))))"+")"
        ws1["F% d" %  rw].value=formula

    row_stamp_2=row_to_write
    row_to_write +=len(factor_title) +1

    #波动率
    ws1["A% d" %  row_to_write].value='波动率'
    ws1["C% d" %  row_to_write].value='均值的绝对值'
    ws1["D% d" %  row_to_write].value='标准差'
    ws1["E% d" %  row_to_write].value='标准差/(均值的绝对值)'
    ws1["F% d" %  row_to_write].value='得分'
    row_to_write +=1

    write_dataframe_to_excel(ws1,factor_title,start_col=1,start_
row=row_to_write,index=False,columns=False)
    for i in range(len(factor_title)):
        rw=row_to_write +i
        formula="=IF(ABS(AVERAGE(C"+str(i+2)+":E"+str(i+2)+"))=0,
0.0001,ABS(AVERAGE(C"+str(i+2)+":E"+str(i+2)+")))"
```

```
            ws1["C% d" % rw].value=formula
        for i in range(len(factor_title)):
            rw=row_to_write +i
            formula="=STDEV(C"+str(i+2)+":E"+str(i+2)+")"
            ws1["D% d" % rw].value=formula
        for i in range(len(factor_title)):
            rw=row_to_write +i
            formula="=D"+str(rw)+"/C"+str(rw)
            ws1["E% d" % rw].value=formula
        for i in range(len(factor_title)):
            rw=row_to_write +i
            formula="=IF(波动率!$ C"+str(i+3)+"<波动率!$ L"+str(i+3)+",
IF(E"+str(rw)+"<波动率!$ C"+str(i+3)+",0,IF(E"+str(rw)+">波动率!
$ L"+str(i+3)+",100,OFFSET(波动率!$ B$ 2,0,MATCH(E"+str(rw)+",波动
率!$ C"+str(i+3)+":$ L"+str(i+3)+")))) "+",IF(E"+str(rw)+">波动率!
$ C"+str(i+3)+",0,IF(E"+str(rw)+"<波动率!$ L"+str(i+3)+",100,
OFFSET(波动率!$ B$ 2,0,MATCH(E"+str(rw)+",波动率!$ C"+str(i+3)+":
$ L"+str(i+3)+",-1))))"+")"
            ws1["F% d" % rw].value=formula

    row_stamp_3=row_to_write
    row_to_write +=len(factor_title) +1


    ##指标数据得分表
    n=int(len(boundarys)/3)
    coefficient_list=importance.coefficients
    order_list=[]
    f_names=list(factors_set)
    f_names=f_names[2:(2+len(factor_title))]

    for i in range(len(f_names)):
        for j in range(len(dictionary.排序)):
            if f_names[i] ==dictionary.iloc[j,2]:
                order_list.append(dictionary.iloc[j,3])

    order_list=order_list +order_list +['0' for i in range(len(f_
names))]

    for i in range(len(boundarys)):
        if (coefficient_list[i] <=0 and order_list[i] =='0') or order
_list[i] =='-1':
```

```
        boundary_temp=[]
        for j in range(9,-1,-1):
            boundary_temp.append(boundarys.iloc[i,j])
        boundarys.iloc[i,:]=boundary_temp

    #同行业#
    ws2=wb.create_sheet('同行业')
    sub_boundary_set=boundarys.iloc[:n,:]
    write_dataframe_to_excel(ws2,factor_title,start_col=1,start_
row=3,index=False,columns=False)
    write_dataframe_to_excel(ws2,sub_boundary_set,start_col=3,
start_row=2,index=False,columns=True)

    #趋势#
    ws3=wb.create_sheet('趋势')
    sub_boundary_set=boundarys.iloc[n:(2*n),:]
    write_dataframe_to_excel(ws3,factor_title,start_col=1,start_
row=3,index=False,columns=False)
    write_dataframe_to_excel(ws3,sub_boundary_set,start_col=3,
start_row=2,index=False,columns=True)

    #波动率#
    ws4=wb.create_sheet('波动率')
    sub_boundary_set=boundarys.iloc[(2*n):,:]
    write_dataframe_to_excel(ws4,factor_title,start_col=1,start_
row=3,index=False,columns=False)
    write_dataframe_to_excel(ws4,sub_boundary_set,start_col=3,
start_row=2,index=False,columns=True)


    ##定量得分
    ws5=wb.create_sheet('定量得分')
    ws5["A2"].value='同行业'
    write_dataframe_to_excel(ws5,factor_title,start_col=2,start_
row=2,index=False,columns=False)
    ws5["A%d" % (n+2)].value='趋势'
    write_dataframe_to_excel(ws5,factor_title,start_col=2,start_
row=n+2,index=False,columns=False)
    ws5["A%d" % (2*n+2)].value='波动率'
    write_dataframe_to_excel(ws5,factor_title,start_col=2,start_
row=2*n+2,index=False,columns=False)
```

```
ws5["D1"].value='权重'
for i in range(len(importance)):
    ws5["D% d" % (i+2)].value=importance.importance[i]

ws5["E1"].value='得分'
for i in range(n):
    rw=row_stamp_1 +i
    formula="=定量评分!F"+str(rw)
    ws5["E% d" % (i+2)].value=formula

    rw=row_stamp_2 +i
    formula="=定量评分!F"+str(rw)
    ws5["E% d" % (n+i+2)].value=formula

    rw=row_stamp_2 +i
    formula="=定量评分!F"+str(rw)
    ws5["E% d" % (2*n+i+2)].value=formula

ws5["F1"].value='加权得分'
for i in range(n*3):
    rw=i +2
    formula="=D"+str(rw)+"*E"+str(rw)
    ws5["F% d" % rw].value=formula

ws5["C% d" % (3*n+2)].value='加权得分'
formula="=SUM(F2:F"+str(n*3+1)+")"
ws5["F% d" % (3*n+2)].value=formula


##定性分析
ws6=wb.create_sheet('定性分析')
ws6["E1"].value='加权得分'
ws6["B2"].value='准则层'
ws6["C2"].value='指标层'

ws6["C3"].value='评估指标'
ws6["D3"].value='符号'
ws6["E3"].value='选项说明'
ws6["F3"].value='选项'
ws6["G3"].value='指标得分'
ws6["H3"].value='指标最终权重'
ws6["I3"].value='最终加权得分'
```

```
ws6["J3"].value='说明'

ws6["A4"].value='机构定性评分 F'
ws6["B4"].value='产业属性 A1'

ws6["C4"].value='经济政策影响'
ws6["D4"].value='P1'
ws6["E4"].value='A.良好的宏观经济环境,国家有政策支持,行业发展可持续增长'
ws6["E5"].value='B.稳定的宏观经济环境,国家有政策支持,行业发展稳定'
ws6["E6"].value='C.比较稳定的宏观经济环境,基本不受政策影响,行业发展有小幅波动'
ws6["E7"].value='D.较为波动的宏观经济环境,受政策影响较大,行业发展有较大幅波动'
ws6["E8"].value='E.较大波动的宏观经济环境,受政策影响很大,行业发展有剧烈波动'

ws6["F4"].value='B'
ws6["G4"].value='=IF(F4="","",VLOOKUP(F4,数据!$A$2:$B$6,2,FALSE))'
ws6["H4"].value='=数据!C22'
ws6["I4"].value='=G4*H4'

ws6["C9"].value='经济周期性'
ws6["D9"].value='P2'
ws6["E9"].value='A.受经济周期的影响很小'
ws6["E10"].value='B.受经济周期的影响较小'
ws6["E11"].value='C.受经济周期的影响一般'
ws6["E12"].value='D.受经济周期的影响较大'
ws6["E13"].value='E.受经济周期的影响很大'

ws6["F9"].value='B'
ws6["G9"].value='=IF(F4="","",VLOOKUP(F4,数据!$A$2:$B$6,2,FALSE))'
ws6["H9"].value='=数据!C22'
ws6["I9"].value='=G4*H4'

ws6["B14"].value='产业属性 A1'

ws6["C14"].value='公司性质'
ws6["D14"].value='P3'
ws6["E14"].value='A.中央国有企业'
```

```
    ws6["E15"].value='B.地方国有企业或公众企业'
    ws6["E16"].value='C.中外合资企业或外商独资企业'
    ws6["E17"].value='D.民营企业'
    ws6["E18"].value='E.其他企业'

    ws6["F14"].value='B'
    ws6["G14"].value='=IF(F14="","",VLOOKUP(F14,数据!$A$2:$B$6,
2,FALSE))'
    ws6["H14"].value='=数据!C33'
    ws6["I14"].value='=G14*H14'

    ws6["C19"].value='已使用授信额度占比'
    ws6["D19"].value='P4'
    ws6["E19"].value='A.<=40% '
    ws6["E20"].value='B.>40% 及<=55% '
    ws6["E21"].value='C.>55% 及<=70% '
    ws6["E22"].value='D.>70% 及<=90% '
    ws6["E23"].value='E.>90% 或无授信'

    ws6["F19"].value='B'
    ws6["G19"].value='=IF(F19="","",VLOOKUP(F19,数据!$A$2:$B$6,
2,FALSE))'
    ws6["H19"].value='=数据!C34'
    ws6["I19"].value='=G19*H19'

    ws6["C24"].value='对外担保余额占净资产的比例'
    ws6["D24"].value='P5'
    ws6["E24"].value='A.<=10% '
    ws6["E25"].value='B.>10% 及<=20% '
    ws6["E26"].value='C.>20% 及<=30% '
    ws6["E27"].value='D.>30% 及<=40% '
    ws6["E28"].value='E.>40% 或无授信'

    ws6["F24"].value='B'
    ws6["G24"].value='=IF(F24="","",VLOOKUP(F24,数据!$A$2:$B$6,
2,FALSE))'
    ws6["H24"].value='=数据!C35'
    ws6["I24"].value='=G24*H24'

    ws6["C29"].value='核心管理层变动'
    ws6["D29"].value='P6'
    ws6["E29"].value='A.近3年没有发生法人变更'
```

```
ws6["E30"].value='B.近 3 年有发生法人变更'

ws6["F29"].value='B'
ws6["G29"].value='=IF(F29="","",VLOOKUP(F29,数据!D2:E3,2,
FALSE))'
ws6["H29"].value='=数据!C36'
ws6["I29"].value='=G29*H29'

ws6["C31"].value='近 1 年来被执行和失信被执行次数'
ws6["D31"].value='P7'
ws6["E31"].value='A.0 次'
ws6["E32"].value='B.1 次'
ws6["E33"].value='C.1 次以上'

ws6["F31"].value='B'
ws6["G31"].value='=IF(F31="","",VLOOKUP(F31,数据!G2:H4,2,
FALSE))'
ws6["H31"].value='=数据!C37'
ws6["I31"].value='=G31*H31'

ws6["C34"].value='近 1 年来是否存在被银行冻结存款或起诉欠款'
ws6["D34"].value='P8'
ws6["E34"].value='A.否'
ws6["E35"].value='B.是'

ws6["F34"].value='B'
ws6["G34"].value='=IF(F34="","",VLOOKUP(F34,数据!J2:K3,2,
FALSE))'
ws6["H34"].value='=数据!C38'
ws6["I34"].value='=G34*H34'

ws6["C36"].value='近 1 年来是否存在被股东起诉'
ws6["D36"].value='P9'
ws6["E36"].value='A.否'
ws6["E37"].value='B.是'

ws6["F36"].value='B'
ws6["G36"].value='=IF(F36="","",VLOOKUP(F36,数据!J2:K3,2,
FALSE))'
ws6["H36"].value='=数据!C39'
ws6["I36"].value='=G36*H36'
```

```
    ws6["C38"].value='近 1 年来是否存在被小贷(民间借贷)公司起诉'
    ws6["D38"].value='P10'
    ws6["E38"].value='A.否'
    ws6["E39"].value='B.是'

    ws6["F38"].value='B'
    ws6["G38"].value='==IF(F38="","",VLOOKUP(F38,数据!J2:K3,2,
FALSE))'
    ws6["H38"].value='=数据!C40'
    ws6["I38"].value='=G38*H38'

    ws6["C40"].value='招聘信息'
    ws6["D40"].value='P11'
    ws6["E40"].value='A.近 3 年招聘职位数>25'
    ws6["E41"].value='B.5<近 3 年招聘职位数<=25'
    ws6["E42"].value='C.0<近 3 年招聘职位数<=5'
    ws6["E43"].value='D.近 3 年招聘职位数=0'

    ws6["F40"].value='B'
    ws6["G40"].value='=IF(F40="","",VLOOKUP(F40,数据!M2:N5,2,
FALSE))'
    ws6["H40"].value='=数据!C41'
    ws6["I40"].value='=G40*H40'

    ws6["H44"].value='机构定性得分:'
    ws6["I44"].value='=SUM(I4:I43)'


    ##数据
    ws7=wb.create_sheet('数据')

    final_w=devide_list(final_w,100)

    score_table=pd.DataFrame()
    score_table['指标选项']=["A","B","C","D","E"]
    score_table['得分']=[100,80,60,40,20]
    write_dataframe_to_excel(ws7,score_table,start_col=1,start_row
=1,index=False,columns=True)

    score_table=pd.DataFrame()
    score_table['管理层指标选项']=["A","B"]
    score_table['得分']=[100,50]
```

```
    write_dataframe_to_excel(ws7,score_table,start_col=4,start_row
=1,index=False,columns=True)

    score_table=pd.DataFrame()
    score_table['司法指标选项-1']=["A","B","C"]
    score_table['得分']=[100,50,0]
    write_dataframe_to_excel(ws7,score_table,start_col=7,start_row
=1,index=False,columns=True)

    score_table=pd.DataFrame()
    score_table['司法指标选项-2']=["A","B"]
    score_table['得分']=[100,0]
    write_dataframe_to_excel(ws7,score_table,start_col=10,start_
row=1,index=False,columns=True)

    score_table=pd.DataFrame()
    score_table['招聘指标选项']=["A","B","C","D"]
    score_table['得分']=[100,75,50,25]
    write_dataframe_to_excel(ws7,score_table,start_col=13,start_
row=1,index=False,columns=True)


    ws7["A9"].value='#####目标层---准则层,权重的确定。#####'
    ws7["C10"].value='权重'
    weigth_table=pd.DataFrame()
    weigth_table["a"]=["A1","A2"]
    weigth_table["b"]=["产业属性","公司属性"]
    weigth_table["c"]=["20% ","80% "]
    write_dataframe_to_excel(ws7,weigth_table,start_col=1,start_
row=11,index=False,columns=False)

    ws7["A14"].value='####重要性判别矩阵####'
    input_dir=" D:/Temp/风控模型/AHP-" + industry + "-重要性判断矩阵
1.csv"
    weigth_table=read_csv(input_dir)
    weigth_table=pd.DataFrame(weigth_table,columns=["A1","A2"],
index=["A1","A2"])
    write_dataframe_to_excel(ws7,weigth_table,start_col=1,start_
row=15,index=True,columns=True)


    ws7["A20"].value='#####准则层 A1---指标层 P,权重的确定#####'
```

```
    ws7["C21"].value='最终权重'
    weigth_table=pd.DataFrame()
    weigth_table["a"]=["P1","P2"]
    weigth_table["b"]=["经济政策影响","经济周期性"]
    weigth_table["c"]=final_w[0:2]
    write_dataframe_to_excel(ws7,weigth_table,start_col=1,start_
row=22,index=False,columns=False)

    ws7["A25"].value='####重要性判别矩阵####'
    input_dir=" D:/Temp/风控模型/AHP-" +industry +"-重要性判断矩阵
2.csv"
    weigth_table=read_csv(input_dir)
    weigth_table=pd.DataFrame(weigth_table,columns=["P1","P2"],
index=["P1","P2"])
    write_dataframe_to_excel(ws7,weigth_table,start_col=1,start_
row=26,index=True,columns=True)


    ws7["A31"].value='#####准则层 A2---指标层 P,权重的确定#####'
    ws7["A32"].value='最终权重'
    weigth_table=pd.DataFrame()
    weigth_table["a"]=["P3","P4","P5","P6","P7","P8","P9","P10",
"P11"]
    weigth_table["b"]=["资源储备","销量","对外担保","核心管理层变动",
"近 1 年来被执行和失信被执行次数","近 1 年来是否存在被银行冻结存款或起诉欠
款","近 1 年来是否存在被股东起诉","近 1 年来是否存在被小贷(民间借贷)公司起
诉","招聘信息"]
    weigth_table["c"]=final_w[2:len(final_w)]
    write_dataframe_to_excel(ws7,weigth_table,start_col=1,start_
row=33,index=False,columns=False)

    ws7["A43"].value='####重要性判别矩阵####'
    input_dir=" D:/Temp/风控模型/AHP-" +industry +"-重要性判断矩阵
3.csv"
    weigth_table=read_csv(input_dir)
    weigth_table=pd.DataFrame(weigth_table,columns=["P3","P4",
"P5","P6","P7","P8","P9","P10","P11"],index=["P3","P4","P5","P6",
"P7","P8","P9","P10","P11"])
    write_dataframe_to_excel(ws7,weigth_table,start_col=1,start_
row=44,index=True,columns=True)
```

```
##综合调整及最终评级
ws8=wb.create_sheet('综合调整及最终评级')
title_list=pd.DataFrame()
title_list["a"]=["定量总得分","定性总得分","总得分","初始内部等级","","公司是否上市(上市:Y,未上市:N)"]
write_dataframe_to_excel(ws8,title_list,start_col=2,start_row=1,index=False,columns=False)

ws8["D1"].value='=定量得分!F' +str(len(boundarys)+2)
ws8["D2"].value='=定性分析!I44'
ws8["D3"].value='=0.7*D1+0.3*D2'
ws8["D4"].value='=OFFSET(L201,MATCH(D3,M202:M211,1),0)'
ws8["D6"].value='N'

ws8["D8"].value='如果“是”,请填“Y”,否则填 N'
ws8["E8"].value='向上调整/向下调整'
ws8["F8"].value='最高内部等级调整'
ws8["G8"].value='最低内部等级调整'

ws8["B9"].value='财报的质量'
ws8["C9"].value='标准的无保留意见'
ws8["F9"].value='=IF(OR(C9="财报未经审计",C9="无法发表意见",C9="否定意见",C9="保留意见"),5,IF(AND(OR(C9="保留意见",C9="带强调事项段的无保留意见"),$ D$ 6="Y"),8,""))'
ws8["G9"].value='=IF(AND(C9="标准的无保留意见",$ D$ 6="Y"),3,"")'

ws8["B11"].value='财务披露的及时性'
ws8["C11"].value='可获得最新年报'
ws8["E11"].value='=IF(C11="只能获取 1 年前财报",-2,IF(C11="只能获取 2 年前及更早的财报",-4,""))'

ws8["B13"].value='审计机构的变更'
ws8["C13"].value='过去 1 年发生过财报审计机构变更情况'
ws8["D13"].value='N'
ws8["E13"].value='=IF(D13="Y",-1,"")'

ws8["B15"].value='综合调整及最终评分'
title_list=pd.DataFrame()
title_list["a"]=["经上下调整后的内部等级","经上下、最高等级调整后的内部等级","经上下、最高、最低等级调整后的内部等级","调整后内部等级","内部调整","内部调整原因: ","内部调整后等级","集团/母公司名字","集团/母公司等级
```

```
调整(上调、下调)","经集团/母公司调整后的内部等级","最终内部等级","最终内部评级"]
    write_dataframe_to_excel(ws8,title_list,start_col=3,start_row=15,index=False,columns=False)
    ws8["E15"].value='=IF(D4+SUM(E9:E13)<1,1,IF(D4+SUM(E9:E13)>10,10,D4+SUM(E9:E13)))'
    ws8["F16"].value='=MIN(E15,F9:F13)'
    ws8["G17"].value='=MAX(F16,G9:G13)'
    ws8["G18"].value='=G17'
    ws8["G19"].value=+1
    ws8["G21"].value='=IF((G18+G19)>10,10,IF((G18+G19)<1,1,(G18+G19)))'
    ws8["F22"].value='集团/母公司内部评级'
    ws8["G22"].value='D'
    ws8["G23"].value=-2
    ws8["G24"].value='=IF((G21+G23)>10,10,IF((G21+G23)<1,1,(G21+G23)))'
    ws8["G25"].value='=G24'
    ws8["G26"].value='=OFFSET(H201,MATCH(G25,I202:I211,1),0)'

    #选项
    selection_list=pd.DataFrame()
    selection_list["a"]=["财报未经审计","无法发表意见","否定意见","保留意见","带强调事项段的无保留意见","标准的无保留意见"]
    write_dataframe_to_excel(ws8,selection_list,start_col=3,start_row=37,index=False,columns=False)

    selection_list=pd.DataFrame()
    selection_list["a"]=["可获得最新年报","可获得最新半年报","可获得最新半年报","只能获取2年前及更早的财报"]
    write_dataframe_to_excel(ws8,selection_list,start_col=4,start_row=37,index=False,columns=False)

    ws8["C64"].value='内部评级'
    selection_list=pd.DataFrame()
    selection_list["a"]=[1,2,3,4,5,6,7,8,9,10,11]
    write_dataframe_to_excel(ws8,selection_list,start_col=3,start_row=65,index=False,columns=False)

    ws8["D64"].value='评级违约概率'
    selection_list=pd.DataFrame()
    selection_list["a"]=["0.005%","0.04%","0.13%","0.33%",
```

```
"0.58% ","0.86% ","1.68% ","3.98% ","7.31% ","11.01% ","56.5% "]
    write_dataframe_to_excel(ws8,selection_list,start_col=4,start_
row=65,index=False,columns=False)

    ws8["E64"].value='与 SP 对应的信用等级'
    selection_list=pd.DataFrame()
    selection_list["a"]=["AAA/AA-","A+, A, A-","BBB+, BBB","BBB-",
"BB+","BB","BB-","B+","B","B-","CCC/C"]
    write_dataframe_to_excel(ws8,selection_list,start_col=5,start_
row=65,index=False,columns=False)

    ws8["F64"].value='上调'
    selection_list=pd.DataFrame()
    selection_list["a"]=[0,1,2,3,4,5,6,7,8,9,-1,-2,-3,-4,-5,-6,
-7,-8]
    write_dataframe_to_excel(ws8,selection_list,start_col=6,start_
row=65,index=False,columns=False)

    ws8["G64"].value='下调'
    selection_list=pd.DataFrame()
    selection_list["a"]=[-1,-2,-3,-4,-5,-6,-7,-8]
    write_dataframe_to_excel(ws8,selection_list,start_col=7,start_
row=65,index=False,columns=False)

    ws8["H110"].value='内部等级'
    selection_list=pd.DataFrame()
    selection_list["a"]=["D","C","CC","CCC-","CCC","CCC+","B-",
"B","B+","BB-","BB","BB+","BBB-","BBB","BBB+","A-","A","A+","AA
-","AA","AA+","AAA"]
    write_dataframe_to_excel(ws8,selection_list,start_col=8,start_
row=111,index=False,columns=False)

    ws8["I110"].value='>='
    selection_list=pd.DataFrame()
    selection_list["a"]=[0,0,0,5,10,15,20,25,30,35,40,45,50,55,60,
65,70,75,80,85,90,95]
    write_dataframe_to_excel(ws8,selection_list,start_col=9,start_
row=111,index=False,columns=False)

    ws8["J110"].value='<'
    selection_list=pd.DataFrame()
    selection_list["a"]=[0,0,0,10,15,20,25,30,35,40,45,50,55,60,65,
```

```
70,75,80,85,90,95,100]
    write_dataframe_to_excel(ws8,selection_list,start_col=10,start
_row=111,index=False,columns=False)

    ws8["K111"].value='实质性违约'
    ws8["K112"].value='技术性违约'

    selection_list=pd.DataFrame()
    selection_list["a"]=["Y","N"]
    write_dataframe_to_excel(ws8,selection_list,start_col=3,start_
row=201,index=False,columns=False)

    selection_list=pd.DataFrame()
    selection_list["a"]=[3,2,1,0,-1,-2,-3]
    write_dataframe_to_excel(ws8,selection_list,start_col=4,start_
row=201,index=False,columns=False)

    selection_list=pd.DataFrame()
    selection_list["a"]=["AAA","AA","A","BBB","BB","B","CCC",
"CC","C","D"]
    write_dataframe_to_excel(ws8,selection_list,start_col=5,start_
row=201,index=False,columns=False)

    selection_list=pd.DataFrame()
    selection_list["a"]=[7,6,5,4,3,2,1,0,-1,-2,-3,-4,-5,-6,-7]
    write_dataframe_to_excel(ws8,selection_list,start_col=6,start_
row=201,index=False,columns=False)

    ws8["H201"].value='内部评级标准'
    selection_list=pd.DataFrame()
    selection_list["a"]=["D","C","CC","CCC","B","BB","BBB","BBB
+","A","AA","AAA"]
    write_dataframe_to_excel(ws8,selection_list,start_col=8,start_
row=202,index=False,columns=False)

    ws8["I201"].value='>='
    selection_list=pd.DataFrame()
    selection_list["a"]=[0,1,2,3,4,5,6,7,8,9]
    write_dataframe_to_excel(ws8,selection_list,start_col=9,start_
row=202,index=False,columns=False)

    ws8["J201"].value='<'
```

```
    selection_list=pd.DataFrame()
    selection_list["a"]=[1,2,3,4,5,6,7,8,9,10]
    write_dataframe_to_excel(ws8,selection_list,start_col=10,start_
row=202,index=False,columns=False)

    ws8["L201"].value='内部等级'
    selection_list=pd.DataFrame()
    selection_list["a"]=[1,2,3,4,5,6,7,8,9,10]
    write_dataframe_to_excel(ws8,selection_list,start_col=12,start_
row=202,index=False,columns=False)

    ws8["M201"].value='>='
    selection_list=pd.DataFrame()
    selection_list["a"]=[0,10,20,30,40,50,60,72,82,92]
    write_dataframe_to_excel(ws8,selection_list,start_col=13,start_
row=202,index=False,columns=False)

    ws8["N201"].value='<'
    selection_list=pd.DataFrame()
    selection_list["a"]=[10,20,30,40,50,60,72,82,92,100]
    write_dataframe_to_excel(ws8,selection_list,start_col=14,start_
row=202,index=False,columns=False)

    #返回生成的 Excel 版本的评分卡模型
    return wb
    #wb.save(' D:/Temp/风控模型/wbtest.xlsx')


#使用开发的评分卡模型,评估样本公司
def company_scoring_and_ranking(factors_set,boundarys,importance,
dictionary,final_w,qualitative_data):
    output_set=pd.DataFrame()
    weight_list=importance["importance"]
    coefficient_list=importance["coefficients"]
    order_list=[]
    f_num=int((len(factors_set.columns)-3)/3)
    f_names=list(factors_set.columns)
    f_names=f_names[2:(2+f_num)]

    for i in range(len(f_names)):
        for j in range(len(dictionary["排序"])):
            if(f_names[i]==dictionary["字段"][j]):
```

```
                order_list.append(int(dictionary["排序"][j]))

    order_list=order_list +order_list

    for i in range(len(f_names)):
        order_list.append(-1)
    factors_set.index=range(len(factors_set))
    for i in range(len(factors_set["CODE"])):
        #i=10
        score_record=pd.Series()
        score_record["CODE"]=factors_set.ix[i,"CODE"]
        score_record["COMP_NAME"]=factors_set["COMP_NAME"].iloc[i]
        isDefault=list(factors_set["isDefault"])[i]

        scores=[]
        factors=factors_set.iloc[i,2:(len(factors_set.columns)-1)]
        #查询评分表
        for j in range(len(factors)):
            #j=1
            if factors[j] ==None:
                scores.append(0)
            elif factors[j] <boundarys.iloc[j,0]:
                  if order _list [j] ==1 or (order _list [j] ==0 and
coefficient_list[j] <0):
                    scores.append(0)
                else:
                    scores.append(100)
            elif factors[j] >=boundarys.iloc[j,0] and factors[j] <
boundarys.iloc[j,1]:
                  if order _list [j] ==1 or (order _list [j] ==0 and
coefficient_list[j] <0):
                    scores.append(10)
                else:
                    scores.append(90)
            elif factors[j] >=boundarys.iloc[j,1] and factors[j] <
boundarys.iloc[j,2]:
                  if order _list [j] ==1 or (order _list [j] ==0 and
coefficient_list[j] <0):
                    scores.append(20)
                else:
                    scores.append(80)
            elif factors[j] >=boundarys.iloc[j,2] and factors[j] <
```

```
boundarys.iloc[j,3]:
                    if order_list[j] ==1 or (order_list[j] ==0 and
 coefficient_list[j] <0):
                      scores.append(30)
                  else:
                      scores.append(70)
              elif factors[j] >=boundarys.iloc[j,3] and factors[j] <
boundarys.iloc[j,4]:
                    if order_list[j] ==1 or (order_list[j] ==0 and
 coefficient_list[j] <0):
                      scores.append(40)
                  else:
                      scores.append(60)
              elif factors[j] >=boundarys.iloc[j,4] and factors[j] <
boundarys.iloc[j,5]:
                    if order_list[j] ==1 or (order_list[j] ==0 and
 coefficient_list[j] <0):
                      scores.append(50)
                  else:
                      scores.append(50)
              elif factors[j] >=boundarys.iloc[j,5] and factors[j] <
boundarys.iloc[j,6]:
                    if order_list[j] ==1 or (order_list[j] ==0 and
 coefficient_list[j] <0):
                      scores.append(60)
                  else:
                      scores.append(40)
              elif factors[j] >=boundarys.iloc[j,6] and factors[j] <
boundarys.iloc[j,7]:
                    if order_list[j] ==1 or (order_list[j] ==0 and
 coefficient_list[j] <0):
                      scores.append(70)
                  else:
                      scores.append(30)
              elif factors[j] >=boundarys.iloc[j,7] and factors[j] <
boundarys.iloc[j,8]:
                    if order_list[j] ==1 or (order_list[j] ==0 and
 coefficient_list[j] <0):
                      scores.append(80)
                  else:
                      scores.append(20)
              elif factors[j] >=boundarys.iloc[j,8] and factors[j] <
```

```
boundarys.iloc[j,9]:
                if order_list[j] ==1 or (order_list[j] ==0 and
coefficient_list[j] <0):
                    scores.append(90)
                else:
                    scores.append(10)
            else:
                if order_list[j] ==1 or (order_list[j] ==0 and
coefficient_list[j] <0):
                    scores.append(100)
                else:
                    scores.append(0)
        #计算得到定量总得分
        score_record["SCORE_QUANTITATE"]=sum(weight_list * scores)
        #定性部分的得分
        level_list=[[100,0],
                    [100,50],
                    [100,60,30],
                    [100,75,50,25],
                    [100,80,60,40,20]]
        #定性部分的选项
        selection_list=[['A','B'],
                        ['A','B'],
                        ['A','B','C'],
                        ['A','B','C','D'],
                        ['A','B','C','D','E']]

        level_select_list=[5,5,5,5,5,2,3,1,1,1,4]
        #计算得到定性总得分
        factors=qualitative_data[qualitative_data['COMP_NAME']==
str(factors_set.iloc[i,1])]
        if len(factors) ==0:
            score_record["SCORE_QUALITATIVE"]=score_record["SCORE_
QUANTITATE"]
        else:
            factors=factors.iloc[0][2:]
            scores_2=[]
            for j in range(len(factors)):
                #j=1
                level_select=level_select_list[j]
                marking=pd.DataFrame()
                marking["SELECTION"]=selection_list[level_select-1]
```

```
                    marking["SCORE"]=level_list[level_select-1]
                    scores_2.append(list(marking[marking['SELECTION'] ==factors[j]].iloc[0])[1])

            score_sum=0
            for j in range(len(final_w)):
                score_sum +=final_w[j] * scores_2[j]
            #定性部分得分
            score_record["SCORE_QUALITATIVE"]=score_sum/100
            #总得分
            score_record["SCORE_W"]=score_record["SCORE_QUANTITATE"] * 0.7 +score_record["SCORE_QUALITATIVE"] * 0.3
            score_record["IS_DEFAULT"]=int(isDefault)
            output_set=output_set.append(score_record,ignore_index=True)

    output_set=output_set.sort_values(by="SCORE_W",axis=0,ascending=False)
    output_set=pd.DataFrame(output_set,columns=['CODE', 'COMP_NAME','SCORE_QUANTITATE', 'SCORE_QUALITATIVE', 'SCORE_W', 'IS_DEFAULT'])

    return output_set


industry="采矿业"#此处以采矿业举例

#入模指标筛选-定量数据
selected_set=index_select(industry,0.2)

#定性数据
try:
    with con.cursor() as cursor:
        query="SELECT * FROM creditrisk."+industry+";"
        cursor.execute(query)
        result=cursor.fetchall()
finally:
    con.close

pre_qualitative_data=pd.DataFrame(result,columns=['CODE', 'COMP_NAME',"经济政策影响",'经济周期性','公司性质','已使用授信额度占比','对外担保','核心管理层变动','近1年来被执行和失信被执行次数','近1年来是否存在被银行冻结存款或起诉欠款','近1年来是否存在被股东起诉','近1年来是否存在
```

```
被小贷(民间借贷)公司起诉','招聘信息'])

qualitative_data=pd.DataFrame()
qualitative_data=qualitative_data.append(pre_qualitative_data.
iloc[0,])

for i in range(1,len(pre_qualitative_data)):
    if pre_qualitative_data.iloc[i,1] in list(qualitative_data.COMP_
NAME):
        continue
    else:
        qualitative_data=qualitative_data.append(pre_qualitative_
data.iloc[i,])

qualitative_data=pd.DataFrame(qualitative_data, columns=['CODE',
'COMP_NAME',"经济政策影响",'经济周期性','公司性质','已使用授信额度占比',
'对外担保','核心管理层变动','近1年来被执行和失信被执行次数','近1年来是
否存在被银行冻结存款或起诉欠款','近1年来是否存在被股东起诉','近1年来是
否存在被小贷(民间借贷)公司起诉','招聘信息'])


#趋势及波动性计算
factors_set_1=calculate_factors(selected_set,2016)
factors_set_2=calculate_factors(selected_set,2015)
factors_set_3=calculate_factors(selected_set,2014)
factors_set=copy.copy(factors_set_1)
factors_set=factors_set.append(factors_set_2)
factors_set=factors_set.append(factors_set_3)

#权重计算
weight_set=factors_set.iloc[:,2:]

for i in range(len(weight_set)):
    for j in range(weight_set.columns.size):
        if math.isinf(weight_set.iloc[i,j]):
            weight_set.iloc[i,j]=0
#逻辑回归
logistic=LogisticRegression()
logistic.fit(weight_set.iloc[:,:-1],weight_set.isDefault.astype
(int))
intercept=logistic.intercept_[0]
imp=list(logistic.coef_[0])
```

```
logcoef=[]

for i in range(len(imp)):
    logcoef.append(math.log(abs(imp[i]) * 1.3 +imp[i]))

for i in range(len(imp)):
    imp[i]=abs(logcoef[i] - (intercept +min(logcoef)))

total=sum(imp)
for i in range(len(imp)):
    imp[i]=imp[i]/total
#计算定量部分的权重
importance=pd.DataFrame({'name': list(weight_set.iloc[:,:-1]), '
importance': imp, 'coefficients': list(logistic.coef_[0])})

#定性权重
final_w=get_qualitative_weigth(industry)

#评分边界计算
boundary_set=factors_set.iloc[:,2:(factors_set.columns.size -1)]
boundarys=pd.DataFrame()
factor_names=list(boundary_set.columns)

for i in range(boundary_set.columns.size):
    temp_boundarys=quantile(boundary_set[factor_names[i]],11)
    boundarys[factor_names[i]] = temp_boundarys[1:(len(temp_
boundarys)-1)]

boundarys.index=[10,20,30,40,50,60,70,80,90,100]
boundarys=boundarys.T

#数据字典
dictionary=readwb("D:/Temp/风控模型/信用风险数据集市-数据字典.xlsx",
"Sheet1")
dictionary=pd.DataFrame(dictionary[1:len(dictionary)],columns=
dictionary[0])
dictionary=dictionary.iloc[0:131,1:5]

factor_type=''
for i in range(len(dictionary.类型)):
    if dictionary.类型[i] !='None':
        factor_type=dictionary.类型.iloc[i]
```

```
        else:
            dictionary.类型[i]=factor_type
    #生成 Excel 版本的评分卡模型,并保存到计算机
    wb= construct _ scorecard (selected _ set, factors _ set, boundarys,
    importance,dictionary,final_w,industry)
    wb.save(' D:/Temp/风控模型/wbtest.xlsx')
    #评估所有样本,并将结果保存到 Excel 中
    rankings= company_scoring_and_ranking(factors_set_1,boundarys,
    importance,dictionary,final_w,qualitative_data)
    rankings_wb=openpyxl.Workbook()
    ws=rankings_wb.active
    write_dataframe_to_excel(ws,rankings,start_col=1,start_row=1,
    index=False,columns=True)
    rankings_wb.save(' D:/Temp/风控模型/'+industry+'_rankings.xlsx')
```

综上,本节重点介绍的是通过调整模型结构来缓释财务粉饰对评估企业信用风险影响的实用方法。对于深入研究并实操过资本市场财务数据统计分析工作的读者来说,肯定会有异常发现,如盈利状况跟违约状态成正比,即盈利能力越强,违约风险越大。虽然这种正相关的系数很小,但毕竟是大于零的正相关系数,也是严重的异常结果。

面对这种情况普遍存在的现状下,很多科学、专业的模型开发方法基本都是不适用的,作为普通的信用债投资者我们无力改变现状,只能改变信用债投资分析的架构和方法,尽量减少财务粉饰对挖掘企业真实还款能力的影响。此处介绍的方法,是笔者多年来的实践经验总结,笔者也相信有更好的方法,欢迎广大读者来信交流。



附录 A Python 语言基础

1. 安装和配置 Python 集成开发环境

在众多的 Python 集成开发环境(IDE)中,Anaconda 无疑是应用最广泛、最受广大 Python 爱好者使用的 IDE 之一。Anaconda 也是一个包含众多 Python 包的集合,安装 Anaconda 的同时就预先集成了如 Numpy、Scipy、Pandas、Scikit-learn 等 180 多个在数据分析领域常用的包,这对于初学者来说是一个天大福音。

除此之外,Anaconda 还有很多的优点,主要包括可以在安装 Anaconda 的同时自动配置环境变量,无须再为配置复杂的环境变量而苦恼,也可通过 Anaconda 管理工具包、开发环境、Python 版本,从而大大简化工作流程。使用 Anaconda 不仅可以方便地安装、更新、卸载工具包,而且安装时能自动安装相应的依赖包,同时还能达到使用不同的虚拟环境隔离不同项目的要求。

安装 Anaconda 时,首先访问 Anaconda 官网(https://www.anaconda.com/),用鼠标选择 Products,单击 Download,如图 A-1 所示。



图 A-1 Anaconda 官网

进入 Anaconda 下载页面,选择 Windows 版本,并单击 Windows 图标,如图 A-2 所示。

选择集成 Python3.6 版本的 Anaconda 下载,根据自身机器硬件的配置,可选择 64 位版本和 32 位版本。根据笔者自己电脑的硬件配置,本书选择集成 32

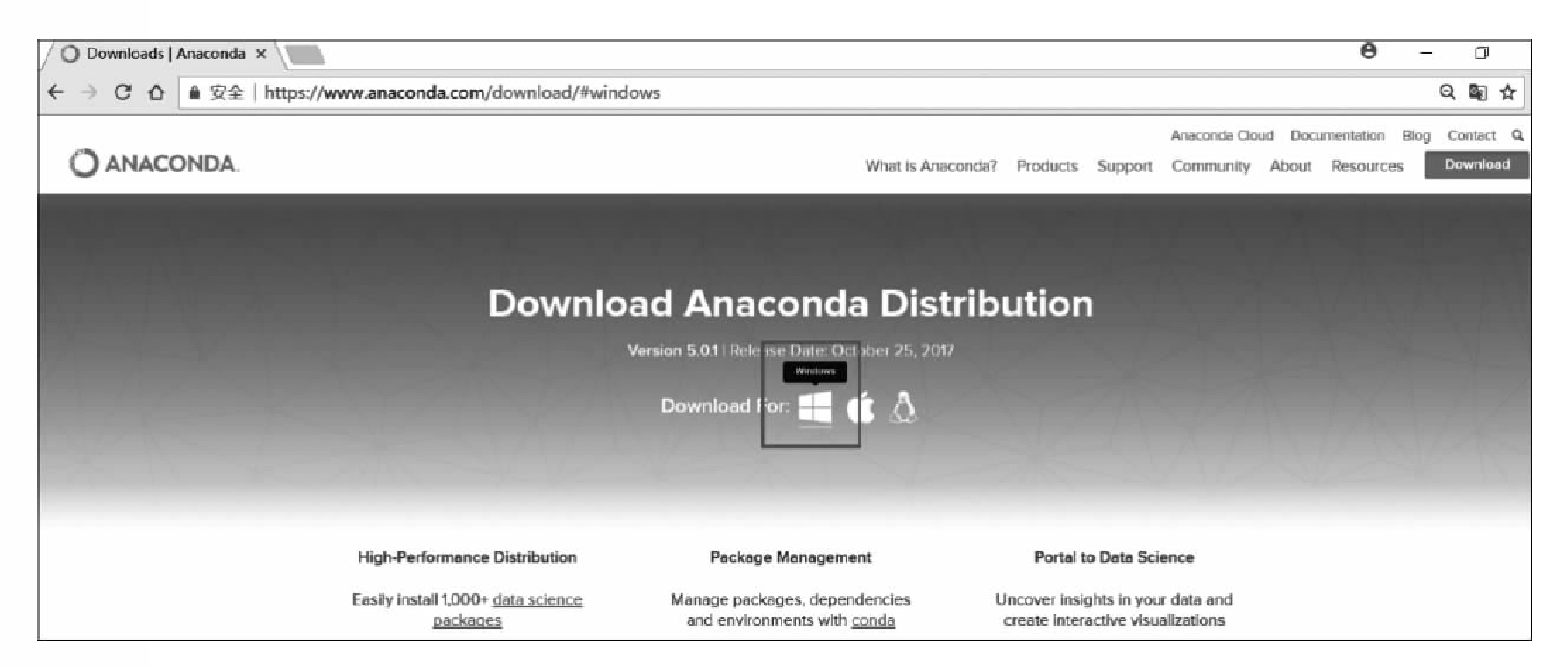


图 A-2 安装 Windows 版本的 Anaconda

位版本的 Anaconda 下载，如图 A-3 所示。

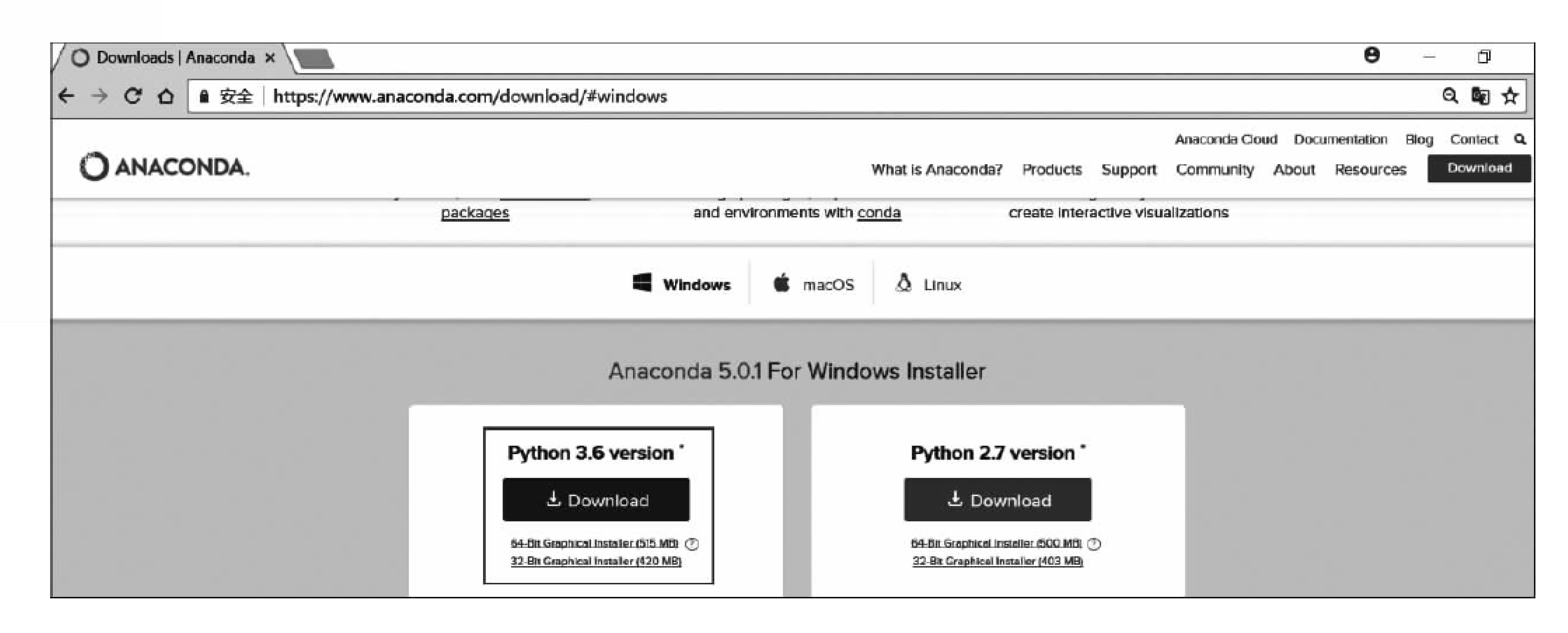


图 A-3 选择 32 位版本的 Anaconda

安装 Anaconda，单击“I Agree”按钮，如图 A-4 所示。

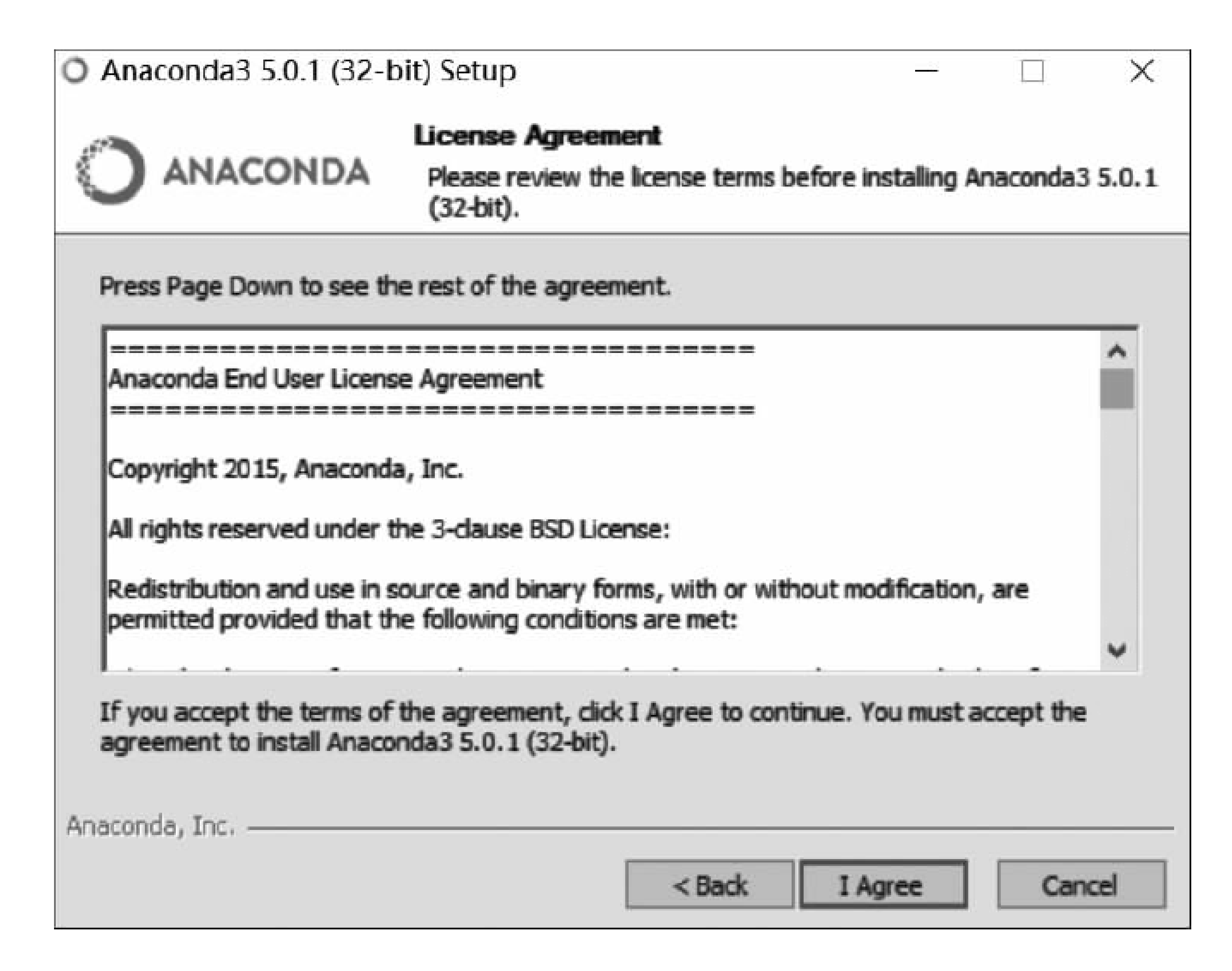


图 A-4 安装 Anaconda

选择 Anaconda 安装的类型，此处我们选择“Just Me(recommended)”，如图 A-5 所示。

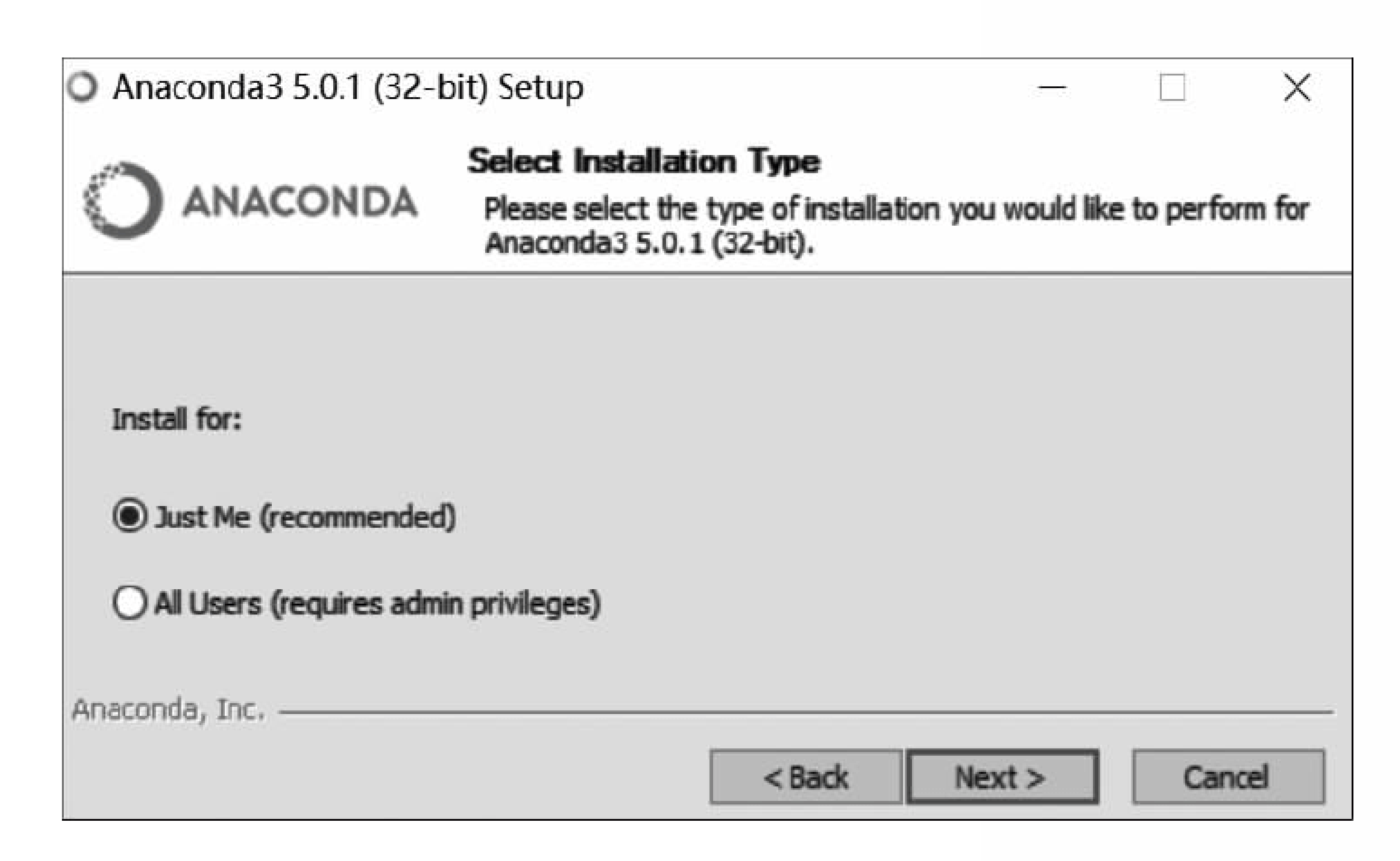


图 A-5　选择 Anaconda 安装类型

选择安装位置，并单击“Next”按钮，如图 A-6 所示。

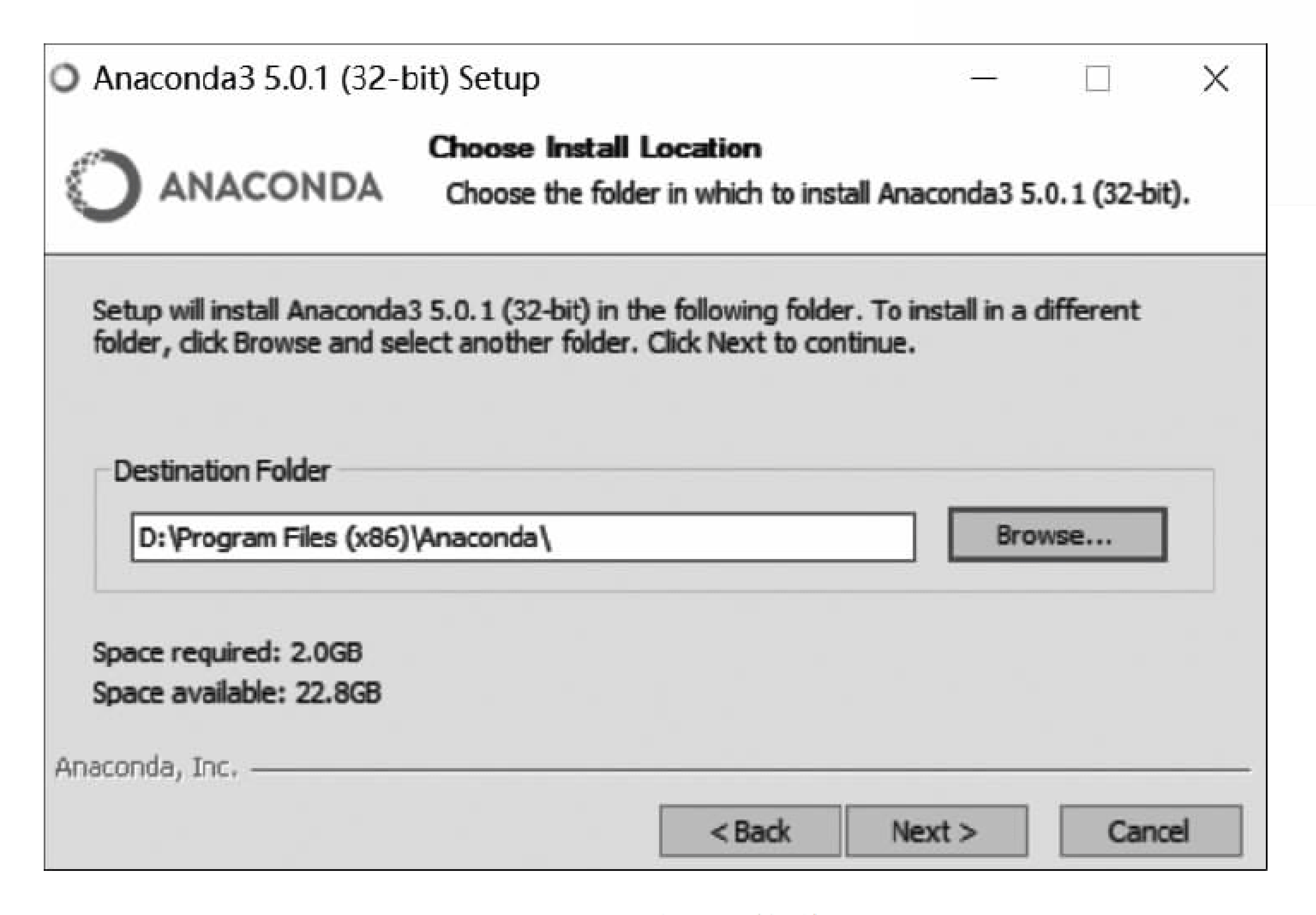


图 A-6　选择安装位置

勾选“Add Anaconda to my PATH environment variable”和“Register Anaconda as my default Python 3.6”两个高级选项，并单击“Install”按钮，如图 A-7 所示。

Anaconda 安装完成，并单击“Next”按钮，如图 A-8 所示。

结束安装，单击“Finish”按钮，如图 A-9 所示。

测试 Python 是否安装成功。单击 Windows 开始，输入“cmd”，并按回车键，在弹出的“命令提示符”中输入“python”，并按回车键。如果显示图 A-10 所示的返回结果，则表明 Anaconda 安装成功。

查看已经安装的 Python 包。重新打开 Windows 命令提示符，输入“conda list”，



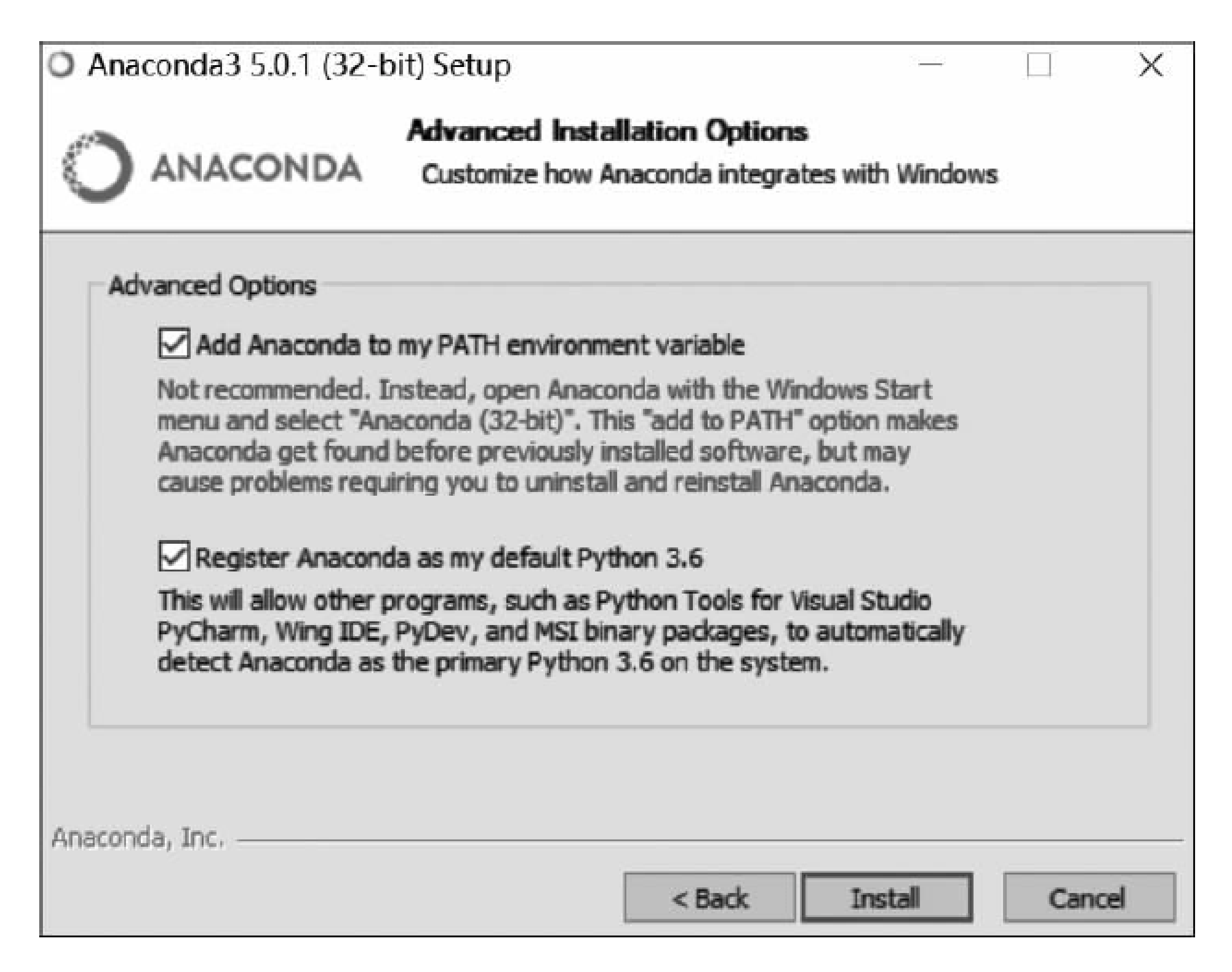


图 A-7　选择安装高级选项

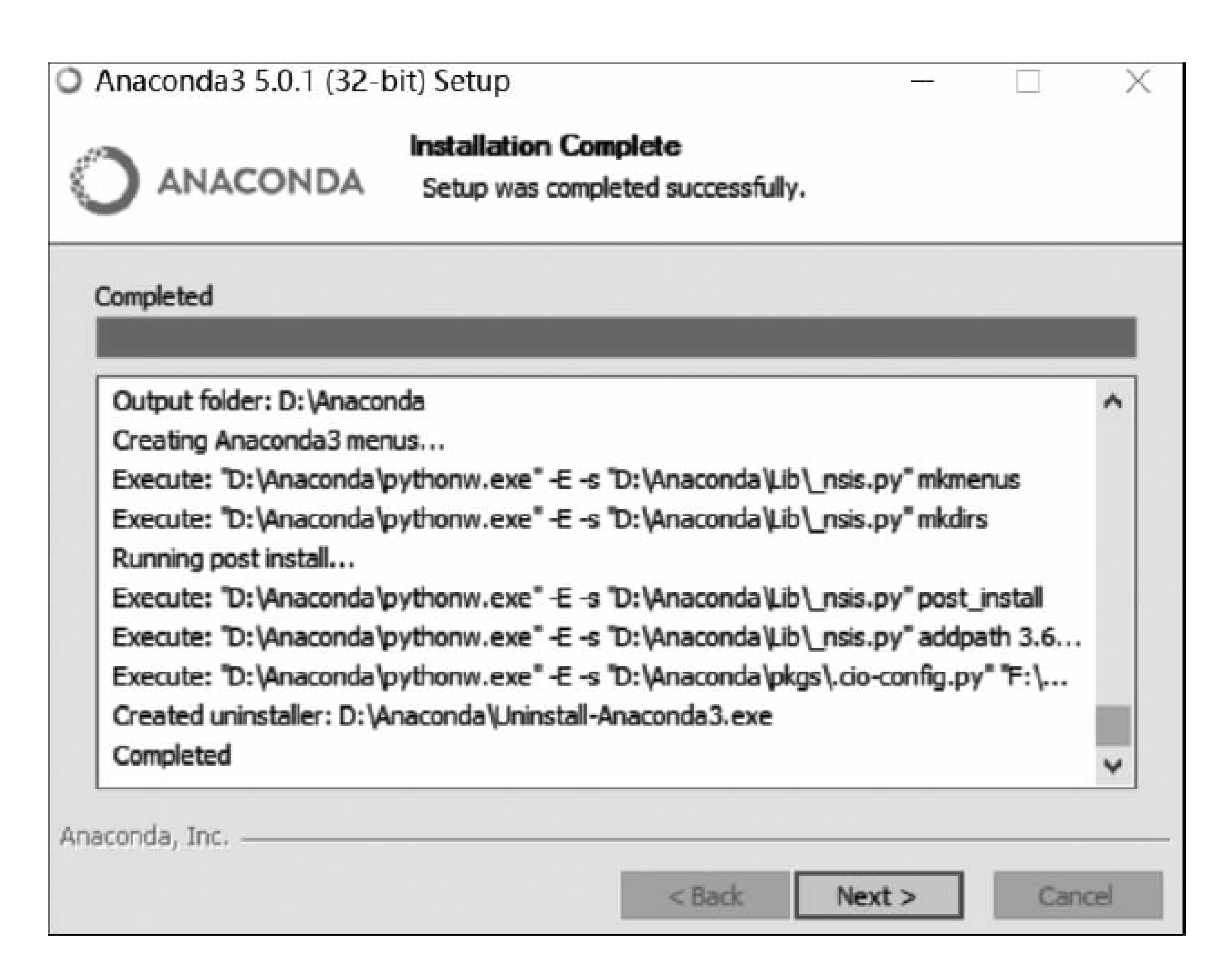


图 A-8　完成安装 Anaconda

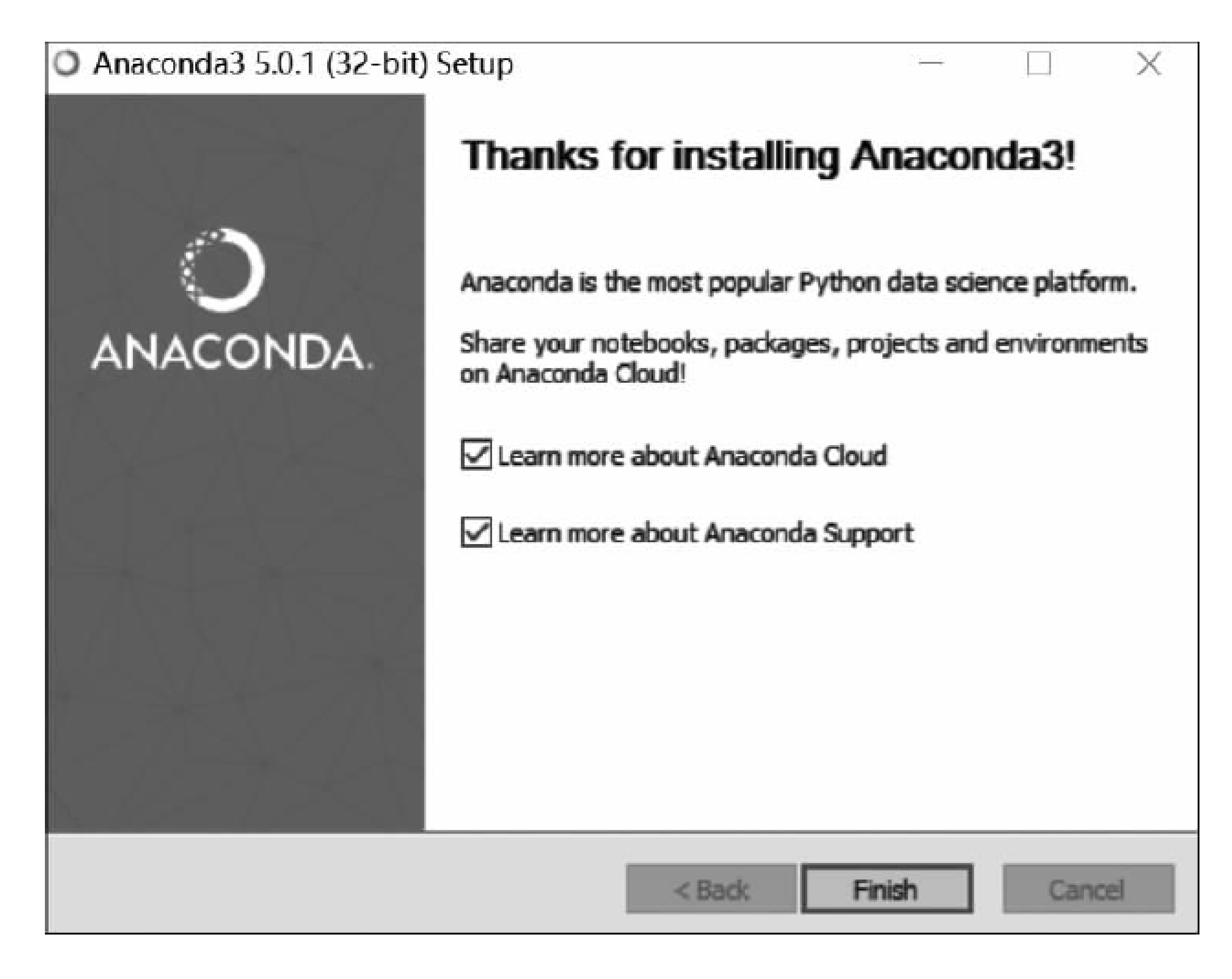


图 A-9　结束安装 Anaconda

```
命令提示符 - python
C:\Users\cuiyu>python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 07:29:16) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

图 A-10　测试 Python 是否安装成功

并按回车键,已经安装的 Python 包会很多,部分 Python 包如图 A-11 所示。

```
命令提示符
Microsoft Windows [版本 10.0.14393]
(c) 2016 Microsoft Corporation。保留所有权利。

C:\Users\cuiyu>conda list
# packages in environment at D:\Anaconda:
#
_ipyw_jlab_nb_ext_conf    0.1.0          py36ha9200a3_0
alabaster                 0.7.10         py36hedafc74_0
anaconda                  5.0.1          py36h2419598_2
anaconda-client           1.6.5          py36hb3b9584_0
anaconda-navigator        1.6.9          py36hfabed4d_0
anaconda-project          0.8.0          py36h88395f3_0
asn1crypto                0.22.0         py36hee29ec9_1
astroid                   1.5.3          py36h3217d1f_0
astropy                   2.0.2          py36h5dd925f_4
babel                     2.5.0          py36h9773feb_0
backports                 1.0            py36hf8a4c8f_1
backports.shutil_get_terminal_size 1.0.0          py36h567543b_2
beautifulsoup4            4.6.0          py36h4361f19_1
bitarray                  0.8.1          py36hc06d7bf_0
bkcharts                  0.2            py36h41dda5e_0
blaze                     0.11.3         py36h70bead1_0
bleach                    2.0.0          py36h2453164_0
bokeh                     0.12.10        py36h6669f50_0
boto                      2.48.0         py36h2dece20_1
bottleneck                1.2.1          py36hd155fb5_0
bzip2                     1.0.6          vc14h31cfb99_1  [vc14]
ca-certificates           2017.08.26          h94faf87_0
cachecontrol              0.12.3         py36h922bf65_0
certifi                   2017.7.27.1    py36h8287e07_0
cffi                      1.10.0         py36h935798a_1
chardet                   3.0.4          py36hf2a9688_1
click                     6.7            py36hfb6b3c4_0
cloudpickle               0.4.0          py36h859eb6c_0
clyent                    1.2.2          py36h4dd741a_1
colorama                  0.3.9          py36he9a059a_0
comtypes                  1.1.2          py36h9d81d3a_0
conda                     4.3.30         py36h404fb56_0
conda-build               3.0.27         py36h3212714_0
conda-env                 2.6.0               h36134e3_1
```

图 A-11　部分 Python 包

测试集成在 Spyder IDE 中的 Python 交互命令 iPython 是否安装成功,打开 Windows 命令行,输入“iPython”并按回车键,输出图 A-12 所示的结果,则表明 iPython 安装成功。

```
选择IPython: C:Users/cuiyu
Microsoft Windows [版本 10.0.14393]
(c) 2016 Microsoft Corporation。保留所有权利。

C:\Users\cuiyu>ipython
Python 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 07:29:16) [MSC v.1900 32 bit (Intel)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

图 A-12　测试 iPython 是否安装成功

单击“开始”,找到 Anaconda3(32-bit),并单击“Spyder”,启动 Python 集成



开发环境 Spyder，如图 A-13 所示。

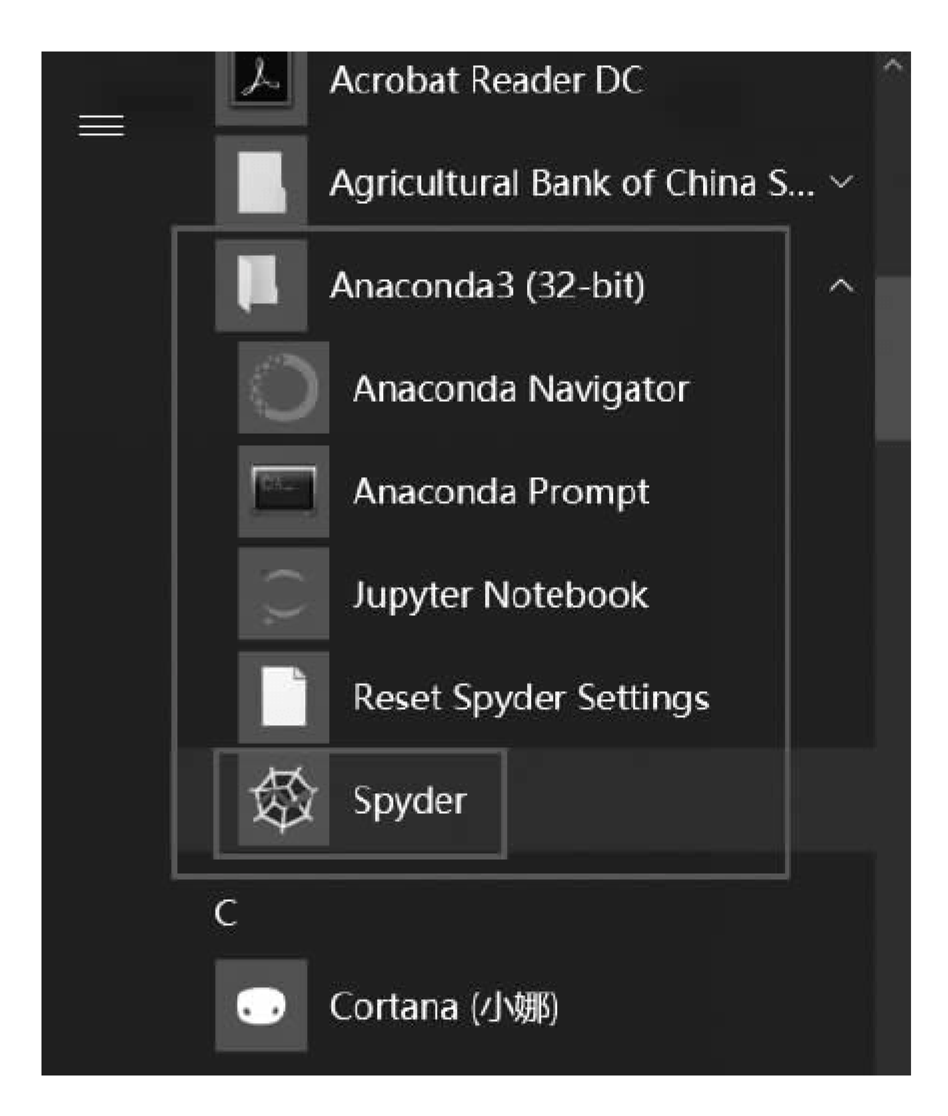


图 A-13　启动 Python IDE Spyder

Spyder 集成开发环境，如图 A-14 所示。

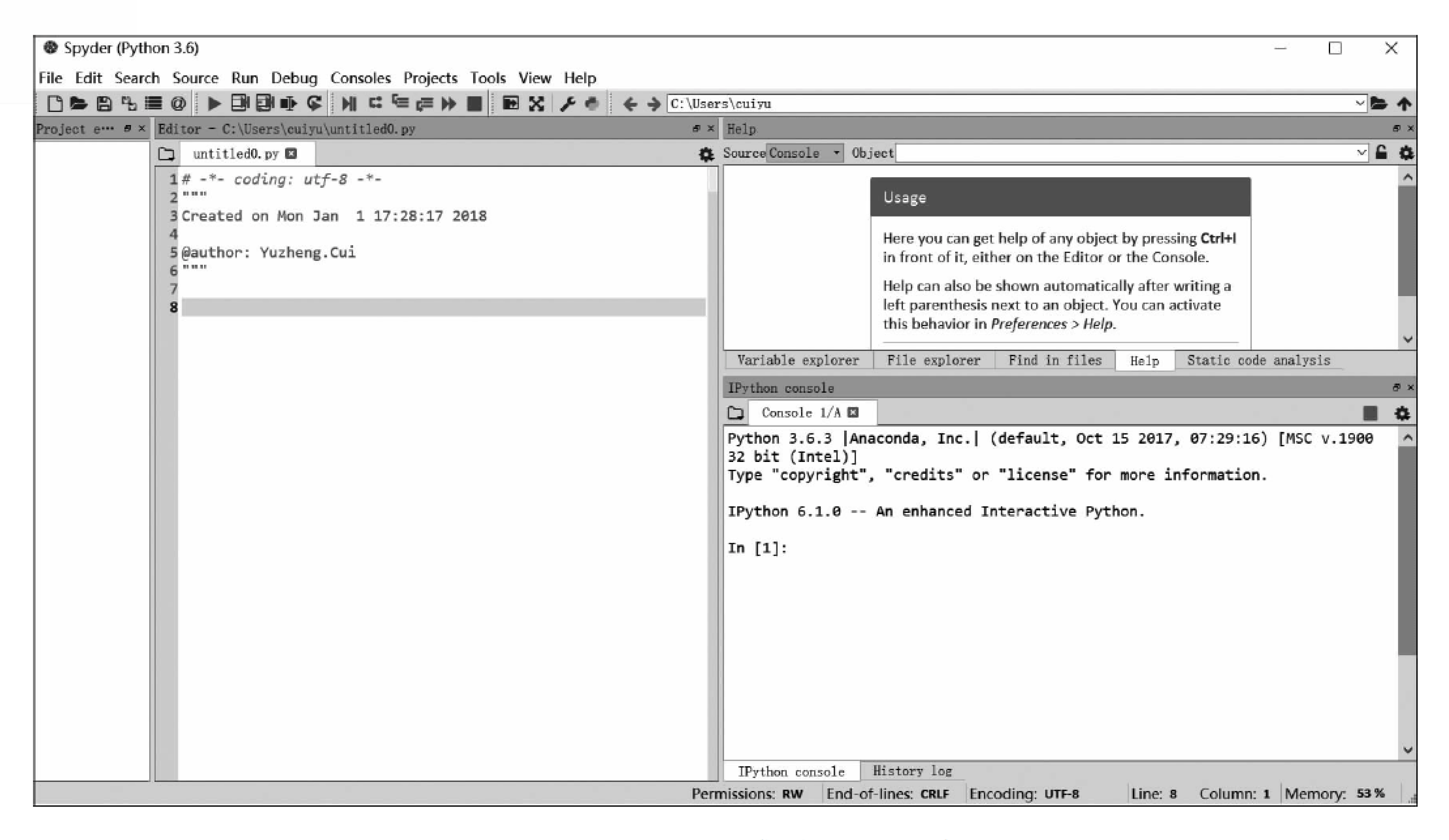


图 A-14　Spyder 集成开发环境

2. 下载 Python 帮助文档

Python 语言集成了大量的关键字和函数，在日常的使用过程中需要经常查阅，为了方便读者在任意环境，尤其是无网络的情况下获取 Python 关键字和函数的帮助，此处介绍一种离线获取帮助的方法。

首先，访问 Python 官网(https://www.python.org/)，并单击“Documentation”，

如图 A-15 所示。

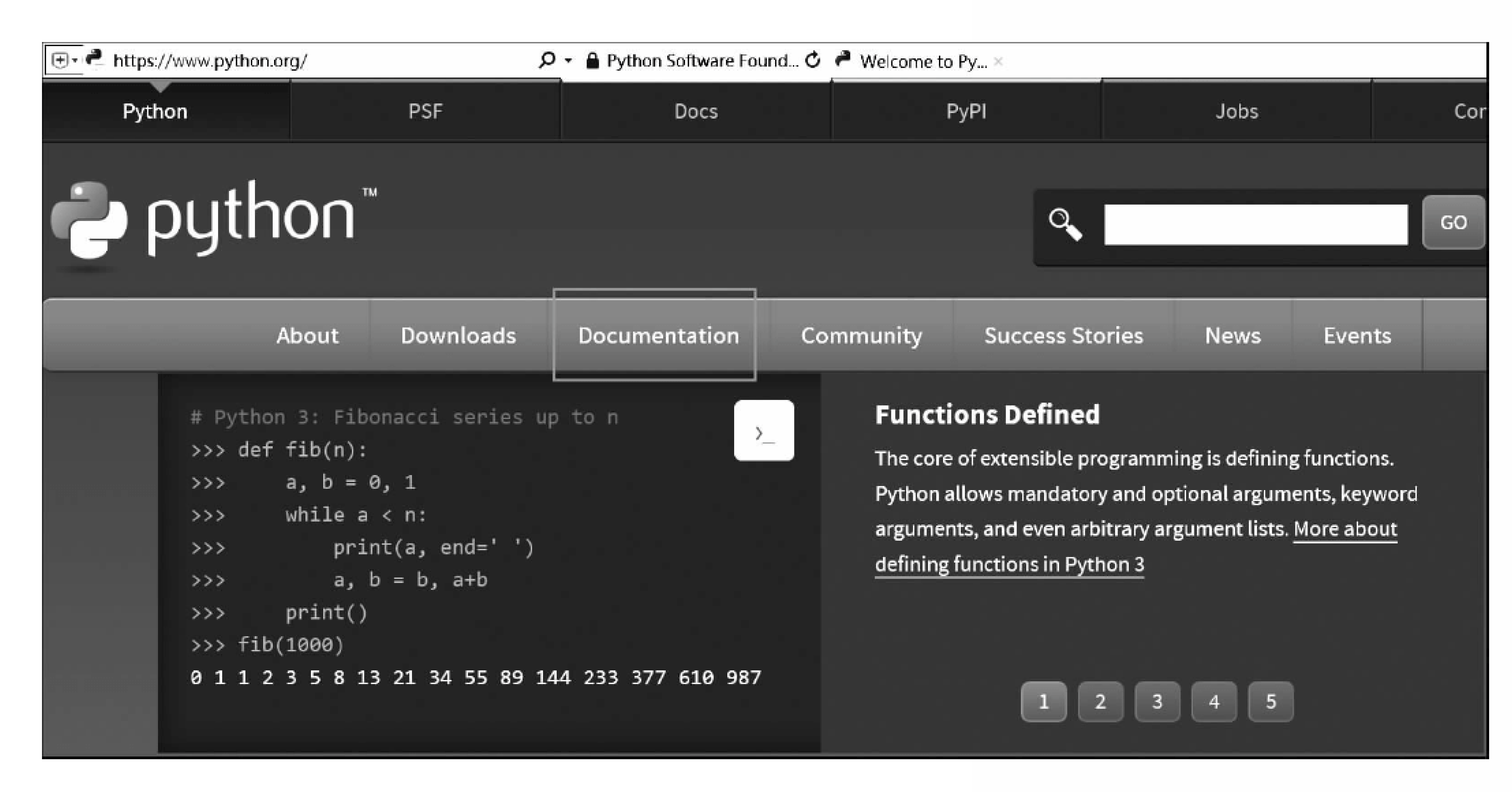


图 A-15　Python 官网

选择下载 Python 3 的帮助文档，单击“Python 3. x Docs”，如图 A-16 所示。

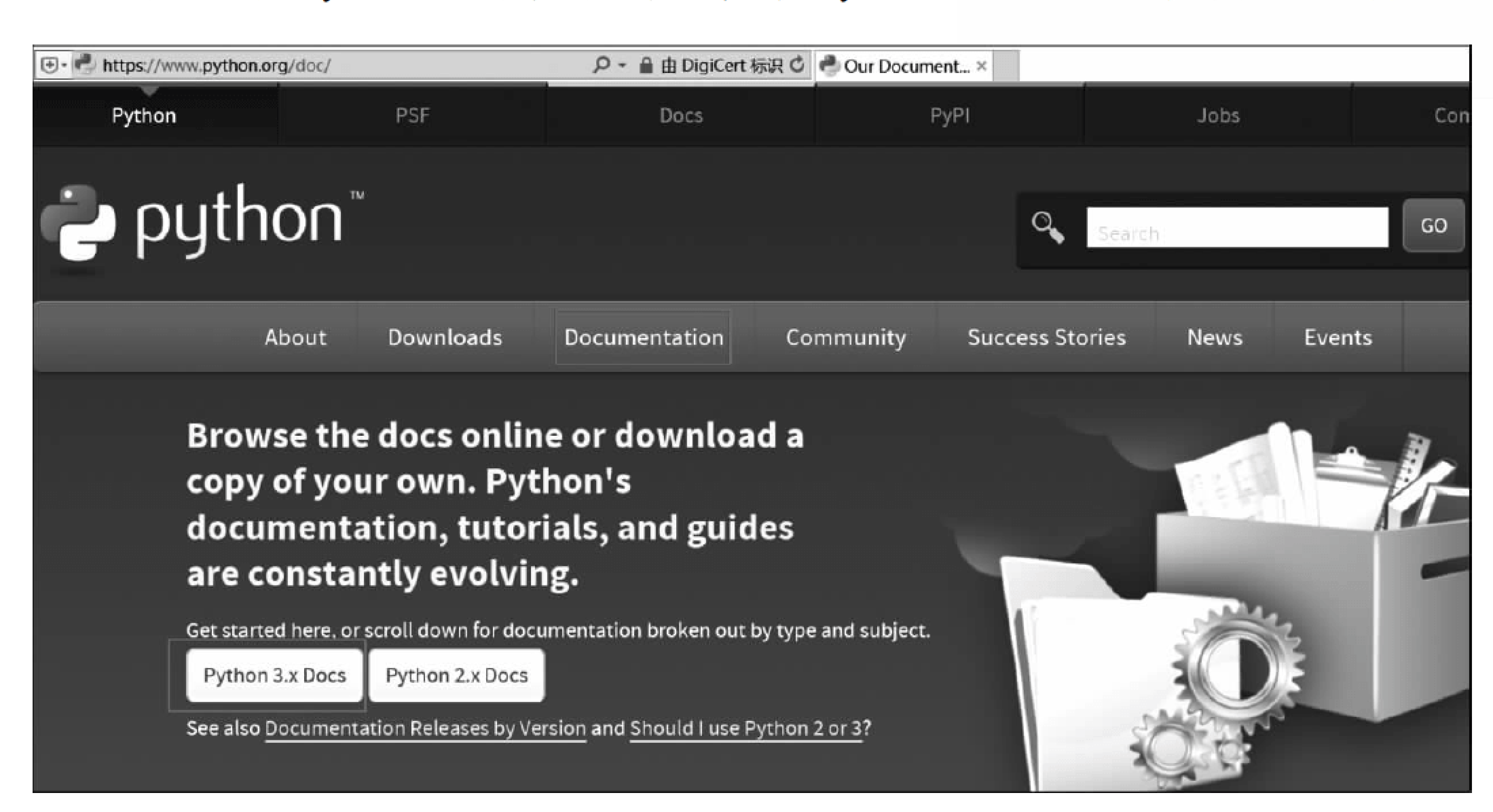


图 A-16　下载 Python 3. x Docs 文档

下载 Python 3. 6. 4 的离线帮助文档，单击“Download these documents”，如图 A-17 所示。

此处，建议下载“HTML”格式的帮助文档，因为其提供的搜索功能，可更全面、更强大地获取读者所需要的帮助信息。右击“Download”，下载“HTML”格式的帮助文档，如图 A-18 所示。

将下载完成的帮助文档，解压到指定文件夹。在所有的解压文件中，找到





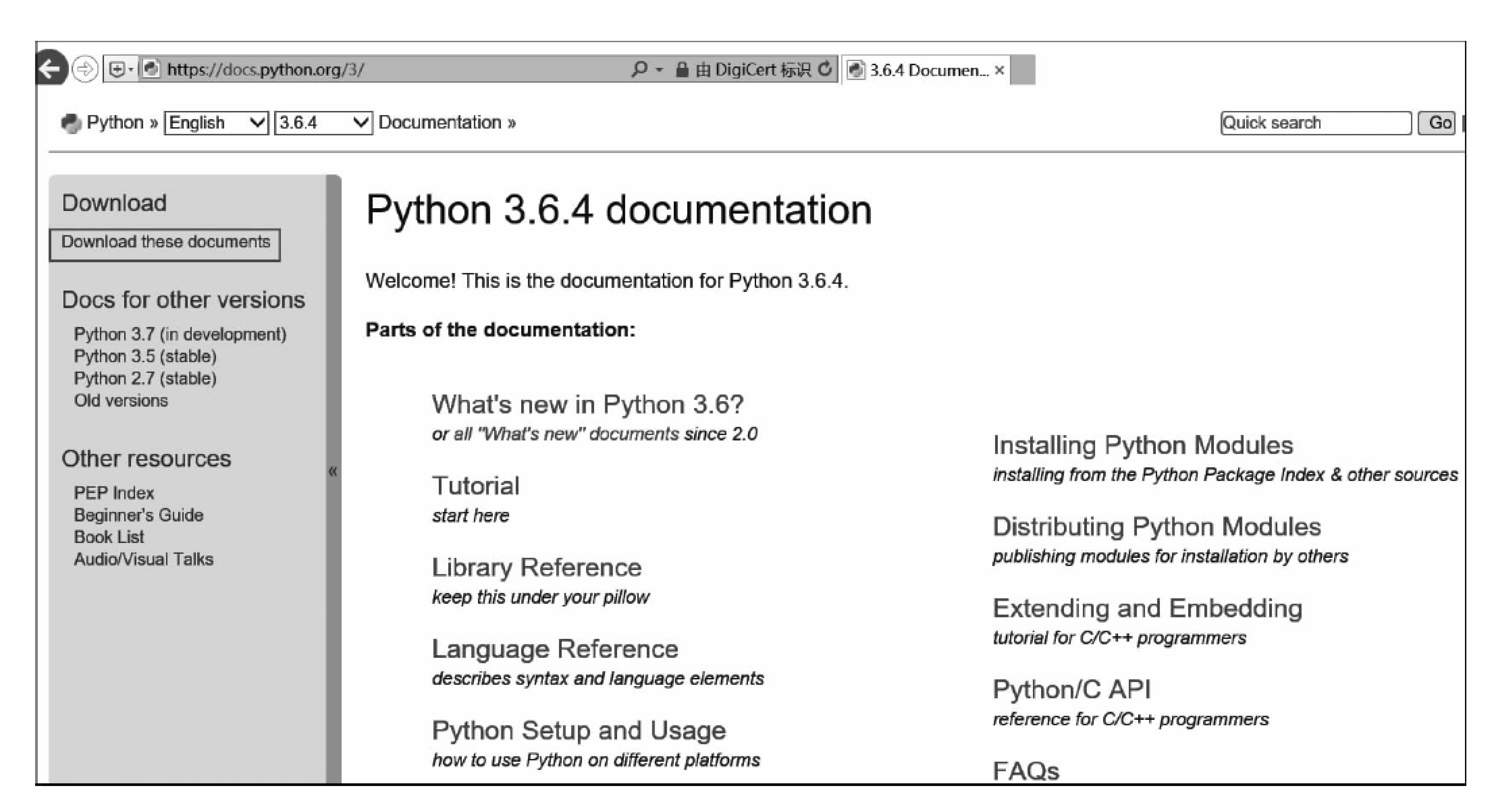


图 A-17　Python 3.6.4 下载页面

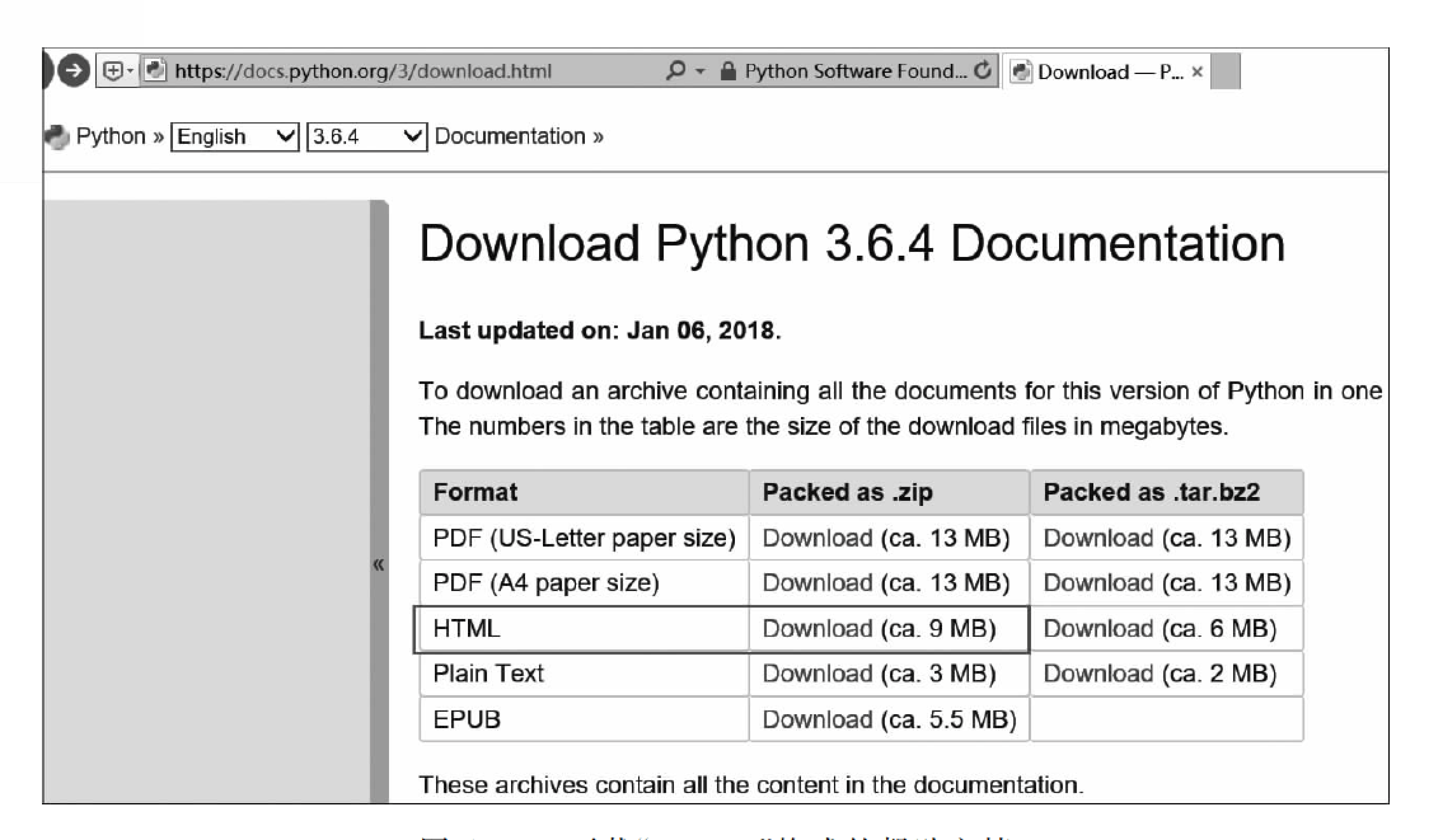


图 A-18　下载“HTML”格式的帮助文档

“index.html”文件，并双击打开，如图 A-19 所示。

如需查找某一函数或关键字的用法，可在右上角的搜索框中输入该关键字，并单击“Go”按钮，如图 A-20 所示。

此处，以获取 Python 内置函数“open”的帮助为例，在图 A-21 所示的搜索结果中，找到含有“Built－in Functions”注释的“open”函数，如图 A-21 所示。

单击图 A-21 中的“open”函数链接，可得到图 A-22 所示的“open”函数的详细语法和参数解释。

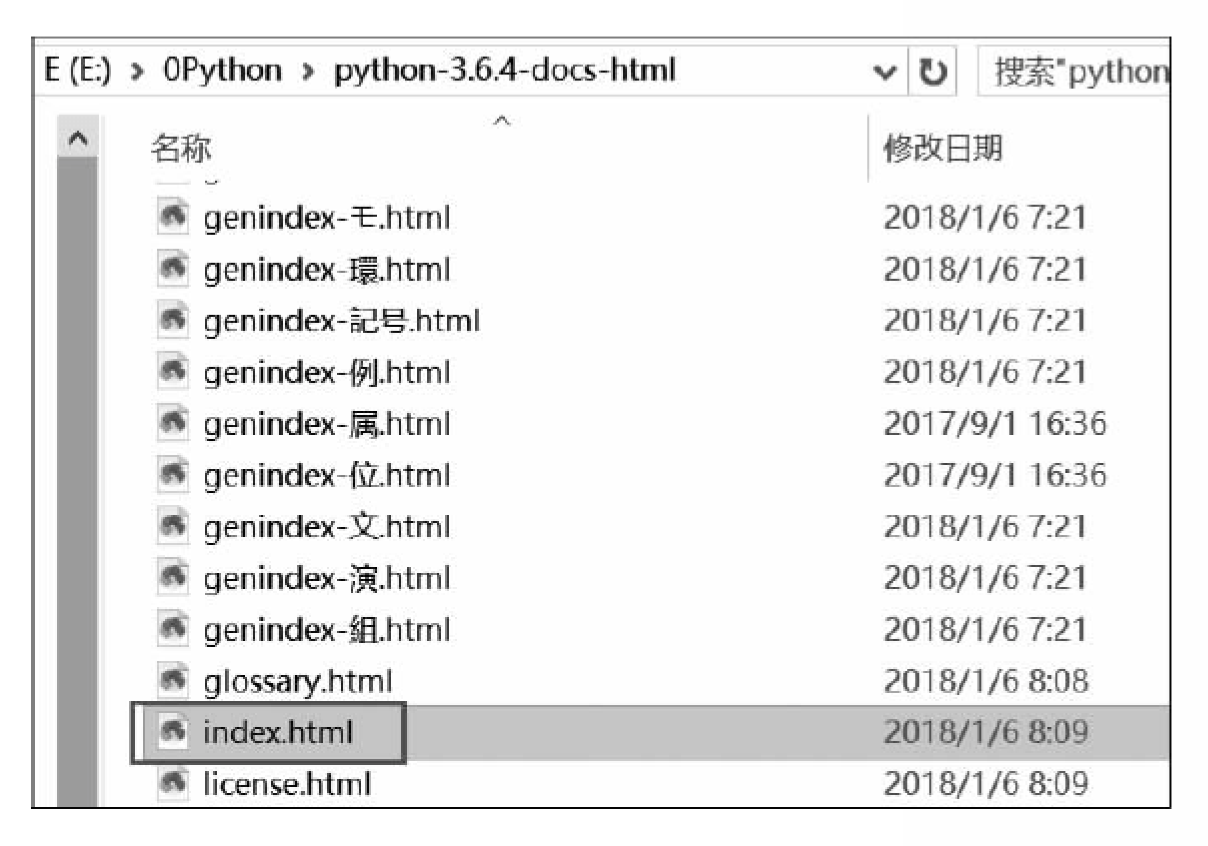


图 A-19　打开“index.html”文件

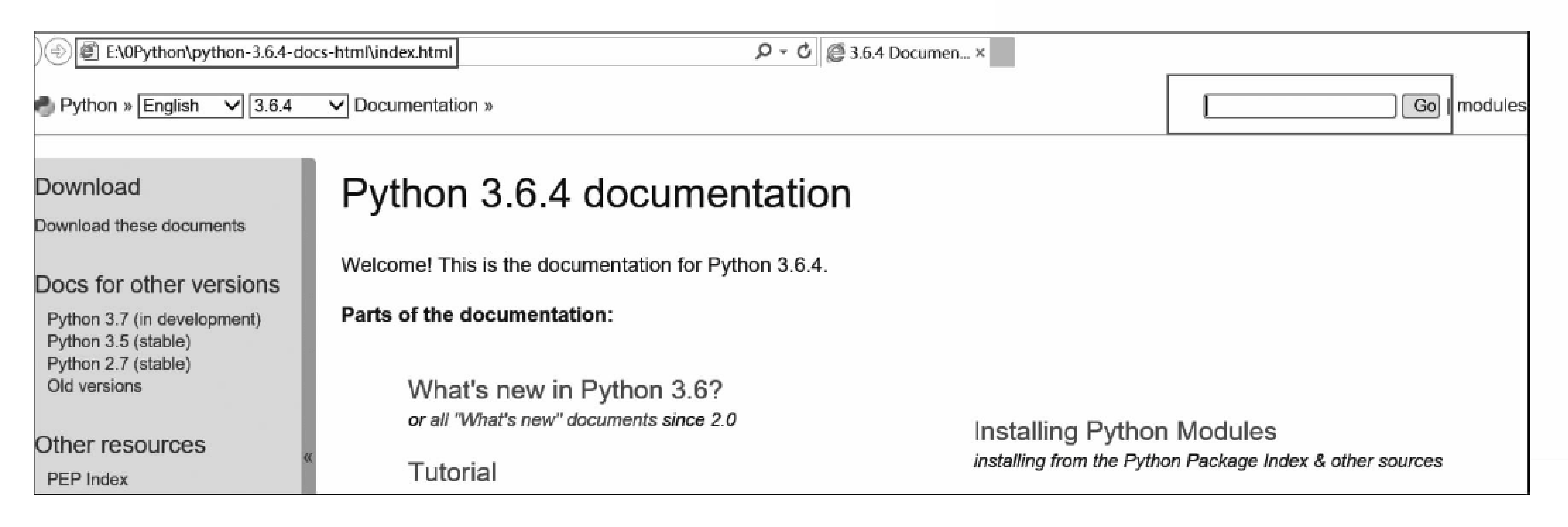


图 A-20　搜索需要帮助的内容

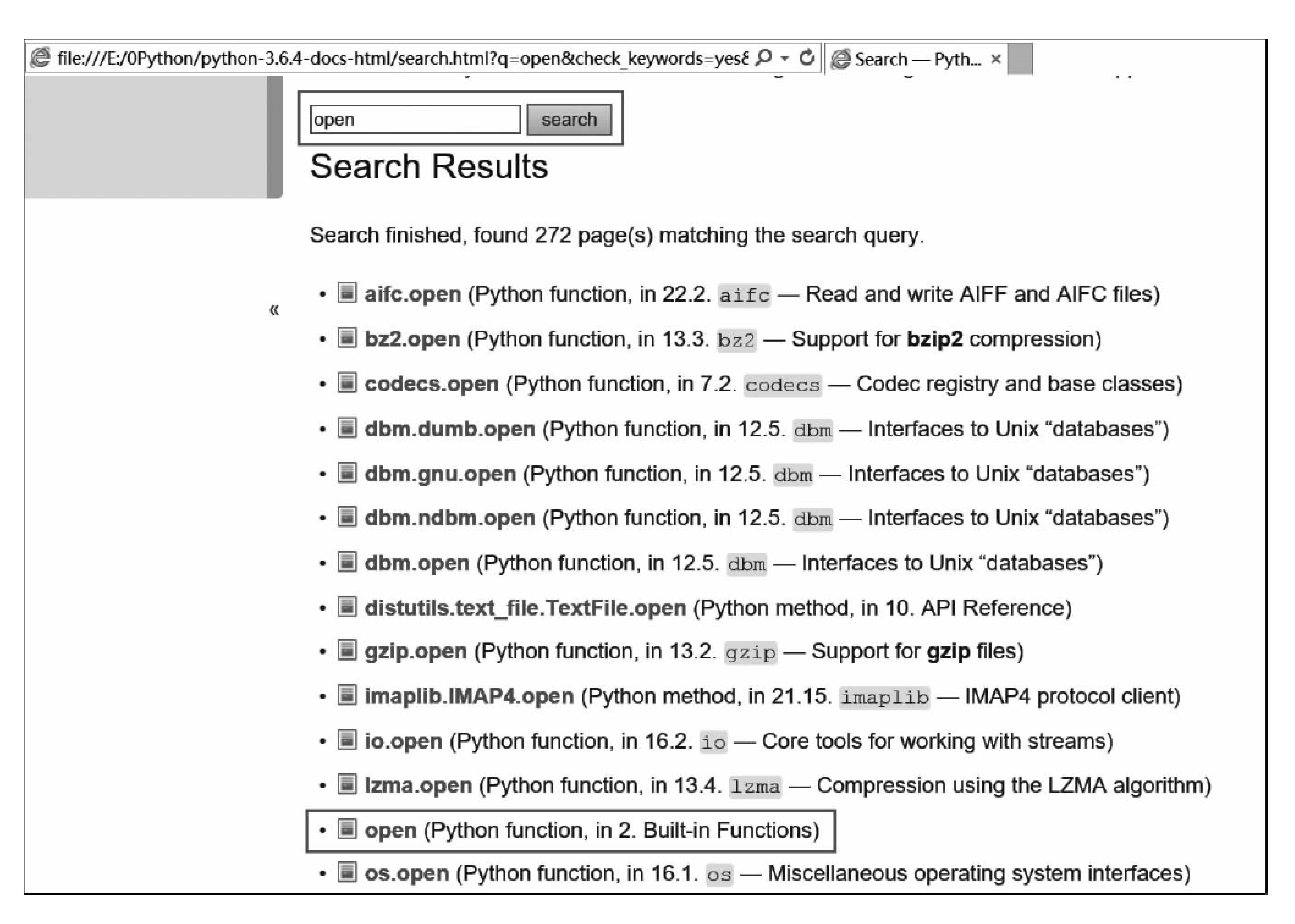
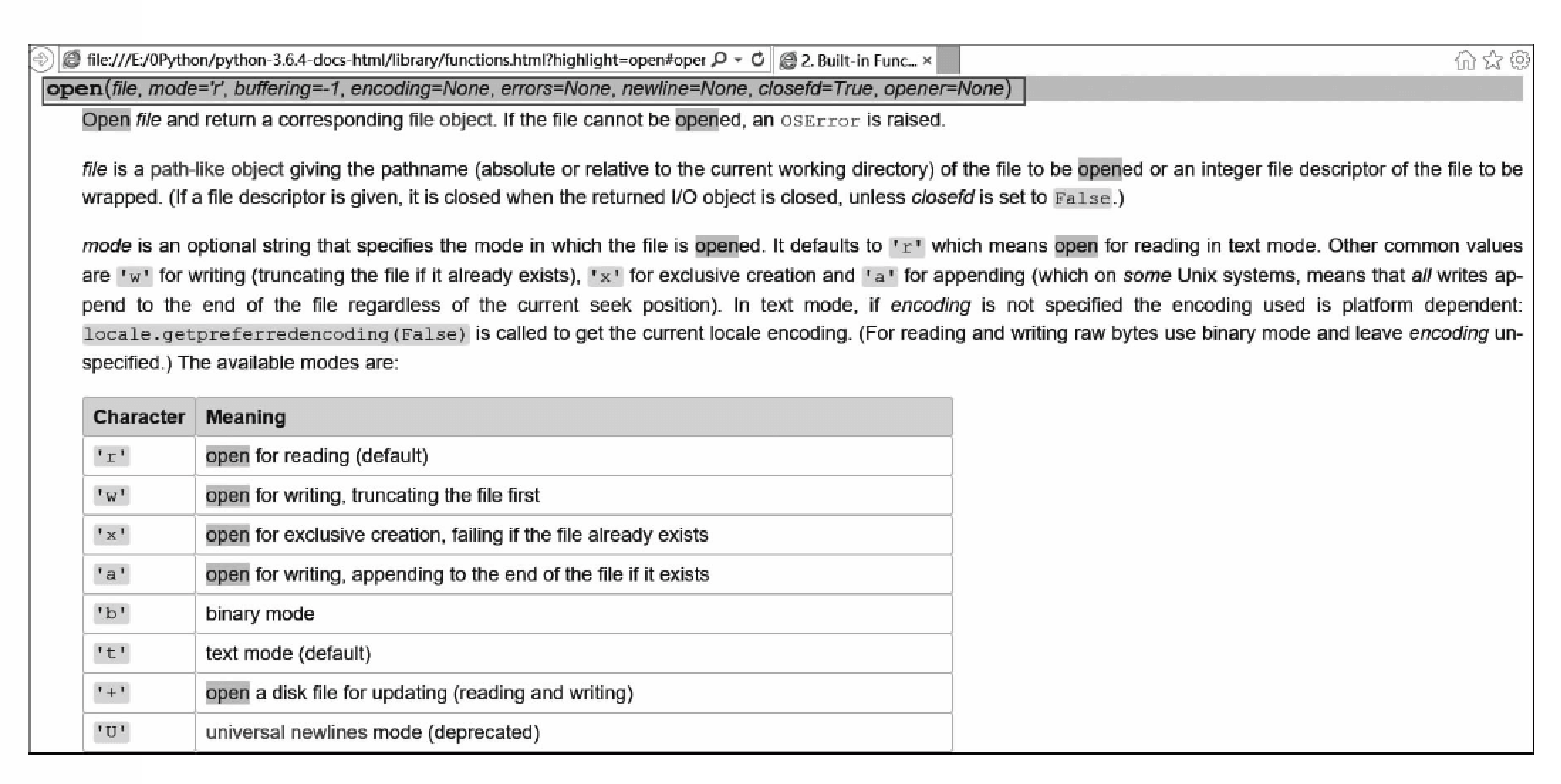


图 A-21　搜索文件读取函数 open



file:///E:/0Python/python-3.6.4-docs-html/library/functions.html?highlight=open#open　2. Built-in Func...

open(*file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None*)

Open *file* and return a corresponding file object. If the file cannot be opened, an `OSError` is raised.

file is a path-like object giving the pathname (absolute or relative to the current working directory) of the file to be opened or an integer file descriptor of the file to be wrapped. (If a file descriptor is given, it is closed when the returned I/O object is closed, unless *closefd* is set to `False`.)

mode is an optional string that specifies the mode in which the file is opened. It defaults to `'r'` which means open for reading in text mode. Other common values are `'w'` for writing (truncating the file if it already exists), `'x'` for exclusive creation and `'a'` for appending (which on *some* Unix systems, means that *all* writes append to the end of the file regardless of the current seek position). In text mode, if *encoding* is not specified the encoding used is platform dependent: `locale.getpreferredencoding(False)` is called to get the current locale encoding. (For reading and writing raw bytes use binary mode and leave *encoding* unspecified.) The available modes are:

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	universal newlines mode (deprecated)

图 A-22　获取文档读写函数的语法和使用帮助

3. Python 语法基础

1）变量和算术表达式

Python 是一种动态类型的语言，在程序执行过程中，变量名称会被绑定到不同的值，而且这些值可以属于不同的类型。赋值运算符的作用仅仅是在名称和值之间创建一种关联，尽管每个值都有一个相关的类型，如 integer 或 string，但变量名称是无类型的。一个变量名称可以在执行过程中引用任意类型的数据。

Python 的语法规则跟 R 语言有很大的不同，采用的是 Pascal 语法规则，即以空格调整的代码块作为基本的代码段或函数体。Python 不会指定所需缩进的量，只要在一个代码块中保持一致即可。然而，每个缩进层次使用 4 个空格是最常见的情况，Spyder 集成开发环境默认采用的就是 4 个空格作为缩进层次的。

我们以一个示例来说明 Python 的变量和表达式，示例代码如下所示。

```
#1)变量和算术表达式
#复利的计算
principal=1000 #本金
rate=0.05 #利率
numYears=10 #年数
year=1 #从第 1 年开始
while year<=numYears:
    principal=principal*(1+rate)
    print(year,round(principal,4)) #输出年份和本金(保留四位有效数字)
    print("%3d %0.4f"%(year,principal)) #格式化输出,year 为 3 位整数,
```

```
principal 为 4 位小数
    print("{0:3d} {1:0.4f}".format(year,principal)) #用 format 函数做
格式化输出
    print(format(year,"3d"),format(principal,"0.2f")) #用 format 函
数做格式化输出
    year+=1
```

2）条件语句

Python 语言中使用 if 和 else 两个关键字来实现条件执行，其中 if 和 else 子句的主体是用缩进来表示的，且 else 子句是可选的。当供选择的条件有多个时，通常使用 or、and 和 not 关键字来组成布尔表达式，以实现条件执行。

Python 语言中没有类似 C 语言中的 switch/case 语句来实现多条件选择，但可以使用 elif 语句来实现这个功能。

我们以一个示例来说明 Python 语言的条件语句，示例代码如下所示。

```
#2)条件语句
x=3
y=8
if x<y:
    print("x is less than y")
else:
    print("x is not less than y")
if x<y:
    pass #创建一条空语句,什么也不执行
else:
    print("Program Executing……")
#使用 or、and 和 not 关键字组成布尔表达式
name="Alice"
type="private memory"
age=6
if name=="Alice" and type=="private memory" and not(age<4 or age>8):
    print("Passed!")
else:
    print("Failed!")
#使用 elif 语句做多条件检验
suffix=".jpg"
context=""
if suffix==".htm":
    content="text/html"
    print("The content is:",content)
elif suffix==".jpg":
    content="image/jpeg"
```

```
        print("The content is:",content)
    elif suffix==".png":
        content="image/png"
        print("The content is:",content)
    elif suffix==".txt":
        content="text/txt"
        print("The content is:",content)
    else:
        raise RuntimeError("Unknown content type!")
```

3）文件的输入和输出

Python 语言中内置了 open()函数用于返回一个新的文件对象，通过调用该对象的方法可以执行各种文件操作，如调用该文件对象的 readline()方法可以读取文件中一行的全部内容，包括每行结尾的换行符也能同时读取。使用 readline()方法读至文件结尾时，将返回空字符串。因此，为了读取一个文件的全部内容，需要循环读取每一行，直到 readline()方法返回空值为止。

在用 open()函数打开文件时，可通过传入特定的参数，表明打开文件的方式，如传入“r”参数时表明以只读的形式打开文件；传入“w”参数时表明以可写的形式打开文件；传入“rw”参数时表明以读写的形式打开文件。Python 语言中还可以使用 input()函数实现交互式地读取用户的输入。

我们以一个示例来说明 Python 语言的文件输入和输出，示例代码如下所示。

```
#3)文件的输入和输出
#读取文件内容,并循环输出
f=open("news-today.txt","r",encoding='UTF-8') #返回一个文件对象,文件内容为中文时,用“UTF-8”编码方式,"r"表示以只读的形式打开文件
line=f.readline()#调用 readline()方法,每次读取一行文件内容
while line:
    print(line,end='')#输出读取的内容,并忽略换行符
    line=f.readline()#继续读取文件的下一行,直到 line 值为空
f.close()
#使用 for 循环输出文件内容
for line in open("news-today.txt","r",encoding='UTF-8'):
    print(line,end='\n')#输出读取的内容,并为每行结尾加上换行符
f.close()
#将程序的输出,写入到文件中
file_out=open("out.doc","w")#新建一个文件"out.doc","w"表示以可写的形式打开
principal=1000  #本金
rate=0.05 #利率
```

```
        print("The content is:",content)
    elif suffix==".png":
        content="image/png"
        print("The content is:",content)
    elif suffix==".txt":
        content="text/txt"
        print("The content is:",content)
    else:
        raise RuntimeError("Unknown content type!")
```

3）文件的输入和输出

Python 语言中内置了 open()函数用于返回一个新的文件对象，通过调用该对象的方法可以执行各种文件操作，如调用该文件对象的 readline()方法可以读取文件中一行的全部内容，包括每行结尾的换行符也能同时读取。使用 readline()方法读至文件结尾时，将返回空字符串。因此，为了读取一个文件的全部内容，需要循环读取每一行，直到 readline()方法返回空值为止。

在用 open()函数打开文件时，可通过传入特定的参数，表明打开文件的方式，如传入“r”参数时表明以只读的形式打开文件；传入“w”参数时表明以可写的形式打开文件；传入“rw”参数时表明以读写的形式打开文件。Python 语言中还可以使用 input()函数实现交互式地读取用户的输入。

我们以一个示例来说明 Python 语言的文件输入和输出，示例代码如下所示。

```
#3)文件的输入和输出
#读取文件内容，并循环输出
f=open("news-today.txt","r",encoding='UTF-8') #返回一个文件对象，文件内容为中文时，用“UTF-8”编码方式，"r"表示以只读的形式打开文件
line=f.readline()#调用 readline()方法，每次读取一行文件内容
while line:
    print(line,end='')#输出读取的内容，并忽略换行符
    line=f.readline()#继续读取文件的下一行，直到 line 值为空
f.close()
#使用 for 循环输出文件内容
for line in open("news-today.txt","r",encoding='UTF-8'):
    print(line,end='\n')#输出读取的内容，并为每行结尾加上换行符
f.close()
#将程序的输出，写入到文件中
file_out=open("out.doc","w")#新建一个文件"out.doc","w"表示以可写的形式打开
principal=1000  #本金
rate=0.05 #利率
```

```
numYears=10 #年数
year=1 #从第 1 年开始
while year<=numYears:
    principal=principal * (1+rate)
#   print("%3d %0.4f" %(year,principal),file=file_out) #向文件对象中写入数据
    file_out.write("%3d %0.4f \n" %(year,principal)) #向文件对象中写入数据
    year+=1
file_out.close()
#交互式输入和输出
name=input("Enter your name :")#在 IPython console 中输入名字
print("Your name is ",name)#输出你的名字
```

4）字符串

Python 语言中可用三种方式创建字符串，分别是将字符串放在单引号、双引号或三引号中。创建字符串时，前后使用的引号必须是对应且配对的。其中，两个三引号之间出现的所有文本都被视为字符串的内容，而使用单引号和双引号指定的字符串必须在一个逻辑行上。Python 语言将字符串存储在字符序列中，因此可以使用整数作为索引来提取字符串中的部分字符，且索引从 0 开始。需要连接两个字符串时，可以使用“＋”运算符。如果字符串为纯数字字符，则 Python 不会自动将字符串的内容隐式地解释为数值，但可以使用 int()或 float()函数显式地将字符串转换为数值。如果需要将数值转换为字符串，则需要使用 str()函数。

我们以一个示例来说明 Python 语言的字符串操作，示例代码如下所示。

```
#4)字符串及相关用法
a="Hello Python"
b='Python is groovy'
c=''' Content-type: text/html
<h1>Hello Python</h1>
Click<a href="http://www.python.org">here</a>.
'''
print(c)
d=a[4] #d='o'
e=a[:5] #e='Hello'
f=a[6:] #f='Python'
g=a[3:8] #g='lo Py'
h=a+" This is a test!" #h='Hello Python This is a test!'
#字符串为数字时,加法执行的是字符串拼接
x="37"
```

```
y="86"
z=x+y #z='3786'
#要执行数学计算,首先要使用 int()或 float()等函数将字符串转换为数值
w=int(x)+int(y) #w=123
i="3.14"
j=float(i)#使用 int(i)则会出错
#将数值转换为字符串
s="The value of x is "+str(z)
```

5）列表

Python 语言中的列表是由任意对象组成的序列，把数据放入方括号中就可以创建列表，也可使用 list()函数来创建列表。列表的索引是从 0 开始的整数，使用 append()函数可将新的数值追加到列表末尾，使用 insert()函数可将新的数值插入到列表指定索引的位置。

列表可以包含任意种类的 Python 对象，包括其他列表。嵌套列表中包含的数据需要使用多次索引才能访问到。

我们以一个示例来说明 Python 语言的列表操作，示例代码如下所示。


```
#5)列表及其应用
fruits=["Apple","Banana","Orange","Pear"]
a=fruits[2] #结果为 Orange
fruits[0]="Cherry" #fruits 的第一项改为"Cherry"
fruits.append("Strawberry") #"Strawberry"被添加到了列表末尾
fruits.insert(2,"Blueberry") #在索引为 2 的列表位置,插入"Blueberry"
b=fruits[0:2] #提取子列表(切片),返回['Cherry', 'Banana']
c=fruits[2:] #提取子列表(切片),返回['Blueberry', 'Orange', 'Pear', '
Strawberry']
d=[1,2]+[3,4,5] #结果是[1,2,3,4,5],用+连接列表
e=[] #创建一个空列表
f=list() #创建一个空列表
g=[1,"Dave",3.14,["Make",7,9,[100,101]],10] #列表中的元素可以包含任意
类型的 Python 对象
g[1] #返回 Dave
g[3][2] #返回 9
g[3][3][1] #返回 101
```

6）元组

Python 语言中创建元组非常简单，只需要把一组数值放入圆括号中即可创建元组。我们同样可以创建包含 0 个元素和 1 个元素的元组。和列表一样，也可以使用数字索引来提取元组中的值。然而，更常见的做法是将元组解包为一组变量。

尽管元组支持的大部分操作与列表相同(如索引、切片、连接等),但创建元组后不能修改它的内容(无法替换、删除现有元组中的元素,也无法向现有元组中添加新元素)。这说明需要将元组看成一个由多个部分组成的单一对象,而不是可在其中插入或删除项的不同对象的集合。

我们以一个示例来说明 Python 语言的元组操作,示例代码如下所示。

```
#6)元组及其应用
stock=('GZMT',10000,785.71) #用圆括号创建元组
person=('First Name','Last Name','Phone Number') #用圆括号创建元组
a=() #0 元组(空元组)
b=("name",) #1 元组(注意随后的逗号)
name,shares,price=stock #将元组解包为一组变量
first_name,last_name,phone=person #将元组解包为一组变量
#portfolio.csv 文件中各行的格式为"CODE,SEC_NAME,CLOSE,SHARES"
filename="portfolio.csv"
portfolio=[]
for line in open(filename):
    fields=line.split(",") #将每行划分为一个列表
    code=fields[0] #提取并转换每个字段
    sec_name=fields[1]
    close=float(fields[2])
    shares=int(fields[3])
    stock=(code,sec_name,close,shares) #创建一个元组(code,sec_name,
close,shares)
    portfolio.append(stock) #将记录追加到列表中
portfolio[0] #输出为('600000.SH', '浦发银行', 12.69, 10000)
portfolio[0][0] #输出为'600000.SH'
portfolio[0][2] #输出为 12.69
#获取该投资组合总持仓
total_cash=0.0
for code,sec_name,close,shares in portfolio:
    total_cash +=close * shares
total_cash #输出总持仓
```

7) 集合

Python 语言中集合用于包含一组无序的对象,用 set()函数创建集合。跟列表和元组不同,集合是无序的,也无法通过数字进行索引;此外,集合中的元素也不能重复,如果赋值给集合时存在重复的元素,则只会保存一个。

集合支持一系列标准操作,如并集、交集、差集和对称差集。

我们以一个示例来说明 Python 语言的集合操作,示例代码如下所示。

```
#7)集合及其应用
```

```
t=set([3,5,8,10,18,'y','H']) #输出结果为{3, 5, 8, 'y', 10, 'H', 18}
s=set("Hello Python") #输出结果为{' ', 'H', 'P', 'e', 'h', 'l', 'n',
'o', 't', 'y'}
a=t|s #t 和 s 的并集
b=t&s #t 和 s 的交集
c=t-s #t 和 s 的差集(在 t 中,但不在 s 中)
d=t^s #t 和 s 的并集(在 t 或 s 中,但不会同时出现在两者中)
t.add('x') #添加一项
s.update([8,18,28]) #在 s 中添加多项
s.remove('e') #删除'e'
```

8) 字典

Python 语言中字典是一个关联数组或散列表,其中包含通过键(key)索引的对象。尽管字符串是最常用的键类型,但还可以使用其他的 Python 对象作为键类型,如数值和元组。需要注意的是列表和字典等对象不能作为键类型,因为它们的内容可以发生变化。如果想获取一个字典的所有关键字,则只需要将字典转换为列表即可。

我们以一个示例来说明 Python 语言的字典操作,示例代码如下所示。

```
#8)字典及其应用
stock={"code":"600000.SH","sec_name":"浦发银行","close":12.69,
"shares":10000} #创建字典
stock["code"] #返回 600000.SH
total=stock["close"] * stock["shares"]
stock["shares"]=500 #修改对象
total2=stock["close"] * stock["shares"]
stock["date"]="2018.1.10"
stock #输出{'close': 12.69,'code': '600000.SH','date': '2018.1.10',
'sec_name': '浦发银行','shares': 500}
a={} #创建一个空字典
b=dict() #使用 Python 关键字 dict 创建字典
keywords=list(stock) #获取字典所有关键字
del stock["sec_name"]
stock #输出{'close': 12.69, 'code': '600000.SH',
'date': '2018.1.10', 'shares': 500}
prices={"600519":747.93,"603444":204.74,"603833":139.03,"603180":
137.65,"603160":93.35}
p=0
if "600519" in prices:
    p=prices["600519"]
else:
    p=0.0
```

```
def divide(a,b):
    q=a//b
    r=a-q*b
    return (q,r)
#调用函数
result=remainder(93,20)
quotient,remainder=divide(93,20)
result #结果为 13
quotient #结果为 4
remainder #结果为 13
count=0
def modifyGlobal():
    global count #修改全局变量的值时,需要首先用 global 关键字声明
    count +=1#修改全局变量的值
    return count
modifyGlobal() #返回值为 1,表明全局变量修改成功
```

Python 语言中有一种函数被称为匿名函数或者 lambda 函数,这其实是一种非常简单的函数:仅由单条语句组成,该语句的结果就是函数的返回值。匿名函数是通过 lambda 关键字定义的,这个关键字没有别的含义,仅仅是说明“我们正在声明的是一个匿名函数”。

如下,介绍 3 个匿名函数应用的示例。

```
#匿名函数示例 1
def short_func(x):
    return x * 2
equiv_short_func=lambda x:x * 2
#如下两种调用方法,输出结果相同
result1=short_func(2)
result2=equiv_short_func(2)
#匿名函数示例 2
def apply_to_list(a_list,f):
    return [f(x) for x in a_list]
ints=[3,6,1,9,0,18]
apply_to_list(ints,lambda x:x * 2) #输出结果为: [6, 12, 2, 18, 0, 36]
#匿名函数示例 3
strings=['foo','card','bar','bbbb','cddc']
strings.sort(key=lambda x:len(set(list(x))))
strings #输出结果为:  ['bbbb', 'foo', 'cddc', 'bar', 'card']
```

11) 对象和类

Python 语言中一切都是对象。对象由内部数据和各种方法组成,这些方法

会执行与这些数据相关的各种操作。前述示例中，在处理像字符串和列表这样的内置类型时，已经用到了对象和方法。Python 语言中使用 dir() 函数可列出对象上的可用方法。使用 class 语句可定义新的类，以实现面向对象编程。类定义中使用 def 语句定义新的方法，每个方法中的第一个参数始终指向对象本身，根据约定，该参数的名称为 self。涉及对象属性的所有操作都必须显式地引用 self 变量。以双下划线开始和结束的方法是特殊的方法，如__init__用于创建对象后初始化该对象。

object 类型是所有 Python 类型的根类型。

我们以一个示例来说明 Python 语言的对象和类的操作，示例代码如下所示。

```
#3.11 对象与类
items=[1,45,67,89] #创建一个列表
items.append(67)
items
dir(items) #列出对象 items 中可用的方法
items.__add__([112,113]) #返回[1, 45, 67, 89, 67, 112, 113]
#class 语句定义新的对象类型
class stack(object):
    def __init__(self):
        self.stack=[]
    def push(self,object):
        self.stack.append(object)
    def pop(self):
        return self.stack.pop()
    def length(self):
        return len(self.stack)
s=stack()
s.push("David")
s.push(34)
s.push([1,2,3])
x=s.pop() #返回[1, 2, 3]
y=s.pop() #返回 34
del s #删除对象 s
dir(stack) #从返回的结果中我们可以看出,dir()函数首先列出的是继承自 object
类的一些方法,最后才是此处我们自定义的 push()、pop()和 length()三个方法。
```

12）异常

Python 语言中如果程序运行出现错误，通常会导致程序终止。但是，我们可以使用 try 和 except 语句来捕捉并处理这些可能产生的异常。通常将可能产生异常的代码段放在 try 语句中，这样如果这些代码产生异常，则会将程序的控制权传

递给 except 代码块中的代码。如果没有出现错误，except 代码块中的代码将被忽略。处理完异常后，程序将会继续执行紧跟在最后一个 except 代码块后面的语句。

我们以一个示例来说明 Python 语言的异常操作，示例代码如下所示。

```
#12)异常
#方式 1: 处理异常;输出信息不同,且程序继续执行
try:
    f=open("result.csv","r")
except IOError as e:
    print(e)
print("Program end! \n")#该语句会被执行,程序没中断
#方式 2: 不处理异常;输出信息不同,且程序中断执行
f=open("result.csv","r")
print("Program end! \n")#该语句不会被执行,程序已经中断
```

13）模块

随着程序变得越来越大，为了便于维护，我们通常需要把它分为多个文件。为此，Python 允许把定义放入一个文件中，然后在其他程序和脚本中将其作为模块导入。要创建模块，可将相关的语句和定义放入与模块同名的文件中（注意，该文件的后缀必须是. py）。

要在其他程序中使用该模块，需要使用 import 语句。import 语句创建了一个新的命名空间，并在该命名空间中执行与. py 文件相关的所有语句。要在导入后访问该命名空间的内容，只要使用该模块的名称作为前缀即可。如果要使用不同的名称导入模块，只需要在 import 语句后加上 as 限定符即可。如果要将具体的定义导入到当前的命名空间中，需要使用 from 语句。

与对象一样，使用 dir 函数也可以列出模块的内容。

我们以一个示例来说明 Python 语言模块的操作，示例代码如下所示。

```
#文件:div.py
def divide(a,b):
    q=a//b
    r=a-q*b
    return (q,r)
#13)模块
import div
a,b=div.divide(1879,51)
import div as fo
a,b=fo.divide(1879,51)
dir(div)
dir(fo)
```

附录 B 本书用到的 Python 包简介

1. numpy

numpy 的主要对象是同类型元素的多维数组，这是一个所有的元素都是一种类型、通过一个正整数元组索引的元素表格（通常是元素是数字）。在 numpy 中维度（dimensions）叫作轴（axes），轴的个数叫作秩（rank）。

如下示例中，我们用 arr 表示 numpy 数组对象，同时需要做如下引入。

```
import numpy as np
```

1）数据的输入和输出

np.loadtxt('file.txt')：从文件"file.txt"中导入数据并返回 ndarray 对象

np.genfromtxt('file.csv',delimiter=',')：从文件"file.csv"中导入数据并返回 ndarray 对象

np.savetxt('file.txt',arr,delimiter='')：将数组保存为"file.txt"文件

np.savetxt('file.csv',arr,delimiter=',')：将数组保存为"file.csv"文件

2）创建数组

np.array([1,2,3,4,5,6])：创建一维数组并返回 ndarray 对象

np.array([(1,2,3),(4,5,6)])：创建二维数组并返回 ndarray 对象

np.zeros(3)：创建长度为 3 的一维数组且数组元素均为 0，返回 ndarray 对象

np.ones((3,4))：创建 3×4 的二维数组且数组元素均为 1，返回 ndarray 对象

np.eye(5)：创建一个 5×5 的二维数组且对角线元素均为 1，其他元素均为 0，返回 ndarray 对象

np.linspace(0,100,6)：创建一维数组且其元素为 0 到 100 之间的 6 等分数字，返回 ndarray 对象

np.arange(0,10,3)：创建一维数组且其元素为以 0 为起点，步长为 3，直到小于 10 的所有数值，返回 ndarray 对象

np.full((2,3),8)：创建 2×3 二维数组且其所有元素均为 8，返回 ndarray 对象

np.random.rand(4,5)：创建4×5二维数组且其所有元素为0～1的随机小数，返回ndarray对象

np.random.rand(6,7)*100：创建6×7二维数组且其所有元素为0～100的随机小数，返回ndarray对象

np.random.randint(5,size=(2,3))：创建2×3二维数组且其所有元素为0～4的随机整数，返回ndarray对象

3）获取数组属性

arr.size：返回数组arr中元素的个数

arr.shape：返回数组arr的维数（行数和列数）

arr.dtype：返回数组arr中元素的类型

arr.astype(dtype)：强制转换数组arr中元素的类型为dtype

arr.tolist()：将数组arr强制转换为Python列表

np.info(np.eye)：查看关于np.eye的文档

4）复制、排序和调整数组

np.copy(arr)：复制数组arr，返回ndarray对象

arr.view(dtype)：以指定dtype类型为数组arr每个元素建立视图

arr.sort()：排序数组，返回ndarray对象

arr.sort(axis=0)：按指定的轴排序数组，返回ndarray对象

two_d_arr.flatten()：将二维数组two_d_arr转换为一维数组，返回ndarray对象

arr.T：返回数组arr的转置（行与列互换）

arr.reshape(3,4)：将数组arr调整为3行4列，并不改变数据，返回ndarray对象

arr.resize((5,6))：将数组arr调整为5行6列，空值用0填充，返回ndarray对象

5）增加、删除元素

np.append(arr,values)：为数组arr增加元素values，返回ndarray对象

np.insert(arr,2,values)：为数组arr在索引为2的元素之前插入元素values，返回ndarray对象

np.delete(arr,3,axis=0)：删除数组arr第3行所有的元素，返回ndarray对象

np.delete(arr,4,axis=1)：删除数组arr第4列所有的元素，返回ndarray对象

6）组合与拆分

np.concatenate((arr1,arr2),axis=0)：将数组arr2按行序拼接在数组arr1的末尾，返回ndarray对象

np.concatenate((arr1,arr2),axis=1)：将数组 arr2 按列序拼接在数组 arr1 的右侧,返回 ndarray 对象

np.split(arr,3)：将数组 arr 拆分为 3 个子数组,返回 ndarray 对象

np.hsplit(arr,5)：将数组 arr 从索引为 5 的元素后水平拆分,返回 ndarray 对象

7）数组的索引、切片、子集

arr[5]：返回数组 arr 的索引为 5 的元素

arr[2,5]：返回二维数组 arr 行索引为 2、列索引为 5 的元素

arr[1]=4：将数组 arr 索引为 1 的元素赋值为 4

arr[1,3]=10：将二维数组 arr 行索引为 1、列索引为 3 的元素赋值为 10

arr[0:3]：返回数组 arr 索引从 0 开始的 3 个元素,如果 arr 为二维数组则返回索引从 0 开始的 3 行全部元素

arr[0:3,4]：返回二维数组 arr 中索引从 0 开始的 3 个行中 4 列的所有元素

arr[0:2]：返回数组 arr 索引从 0 开始的 2 个元素,如果 arr 为二维数组则返回索引从 0 开始的 2 行全部元素

arr[:,1]：返回数组 arr 中第 1 列中所有行的元素

arr<5：返回布尔型数组,其中数组 arr 中小于 5 的元素为 True,大于等于 5 的元素为 False

(arr1<3)&(arr2>5)：返回布尔型数组

~arr：返回布尔型数组的逆(True 变为 False、False 变为 True)

arr[arr<5]：返回数组中小于 5 的元素

8）标量数学

np.add(arr,1)：为数组 arr 的每个元素都加 1,并返回 ndarray 对象

np.subtract(arr,2)：为数组 arr 的每个元素都减 2,并返回 ndarray 对象

np.multiply(arr,3)：为数组 arr 的每个元素都乘 3,并返回 ndarray 对象

np.divide(arr,4)：为数组 arr 的每个元素都除 4,并返回 ndarray 对象

np.power(arr,5)：为数组 arr 的每个元素都计算 5 次方,并返回 ndarray 对象

9）向量数学

np.add(arr1,arr2)：将数组 arr1 和 arr2 的对应元素相加,并返回 ndarray 对象

np.subtract(arr1,arr2)：将数组 arr1 和 arr2 的对应元素相减,并返回 ndarray 对象

nu.multiply(arr1,arr2)：将数组 arr1 和 arr2 的对应元素相乘,并返回 ndarray 对象

np. divide(arr1, arr2)：将数组 arr1 和 arr2 的对应元素相除，并返回 ndarray 对象

np. power(arr1,arr2)：将数组 arr2 中的元素作为数组 arr1 中对应元素的指数，并返回 ndarray 对象

np. array_equal(arr1,arr2)：判断数组 arr1 和 arr2 中对应元素是否相同，并返回 ndarray 对象的布尔型数组

np. sqrt(arr)：计算数组 arr 中每个元素的平方根，并返回 ndarray 对象

np. sin(arr)：计算数组 arr 中每个元素的正弦值(sin())，并返回 ndarray 对象

np. log(arr)：计算数组 arr 中每个元素的自然对数，并返回 ndarray 对象

np. abs(arr)：计算数组 arr 中每个元素的绝对值，并返回 ndarray 对象

np. ceil(arr)：对数组 arr 中每个元素向上取整，并返回 ndarray 对象

np. floor(arr)：对数组 arr 中每个元素向下取整，并返回 ndarray 对象

np. round(arr)：对数组 arr 中每个元素四舍五入取整，并返回 ndarray 对象

10）统计数值

np. mean(arr,axis=0)：计算数组 arr 中指定轴的均值，并返回 ndarray 对象

arr. sum()：返回数组 arr 中所有元素的和

arr. min()：返回数组 arr 中最小的元素

arr. max(axis=0)：返回数组 arr 中指定轴的最大元素

np. var(arr)：返回数组 arr 中所有元素的方差

np. std(arr,axis=1)：返回数组 arr 中指定轴的标准差

arr. corrcoef()：返回数组 arr 中所有元素的相关系数

2. Pandas

Pandas 提供了使我们能够快速便捷地处理结构化数据的大量数据结构和函数，其中两个主要数据结构是 DataFrame 和 Series。此处，我们以 df 代表任意的 Pandas DataFrame 对象，s 代表任意的 Pandas Series 对象，同时我们需要做如下的引入：

```
import pandas as pd
import numpy as np
```

1）导入数据

pd. read_csv(filename)：从 CSV 文件导入数据

pd. read_table(filename)：从限定分隔符的文本文件导入数据

pd. read_excel(filename)：从 Excel 文件导入数据

pd.read_sql(query, connection_object):从 SQL 表/库导入数据

pd.read_json(json_string):从 JSON 格式的字符串导入数据

pd.read_html(url):解析 URL、字符串或者 HTML 文件,抽取其中的 tables 表格

pd.read_clipboard():从你的粘贴板获取内容,并传给 read_table()

pd.DataFrame(dict):从字典对象导入数据,Key 是列名,Value 是数据

2) 导出数据

df.to_csv(filename):导出数据到 CSV 文件

df.to_excel(filename):导出数据到 Excel 文件

df.to_sql(table_name, connection_object):导出数据到 SQL 表

df.to_json(filename):以 Json 格式导出数据到文本文件

df.to_html(filename):导出数据到 HTML 文件

df.to_clipboard():导出数据到粘贴板

3) 创建测试对象

pd.DataFrame(np.random.rand(20,5)):创建 20 行 5 列的随机数组成的 DataFrame 对象

pd.Series(my_list):从可迭代对象 my_list 创建一个 Series 对象

df.index = pd.date_range('1900/1/30', periods=df.shape[0]):增加一个日期索引

4) 查看、检查数据

df.head(n):查看 DataFrame 对象的前 n 行

df.tail(n):查看 DataFrame 对象的最后 n 行

df.shape():查看行数和列数

df.info():查看索引、数据类型和内存信息

df.describe():查看数值型列的汇总统计

s.value_counts(dropna=False):查看 Series 对象的唯一值和计数

df.apply(pd.Series.value_counts):查看 DataFrame 对象中每一列的唯一值和计数

5) 数据选取

df[col]:根据列名,并以 Series 的形式返回列

df[[col1, col2]]:以 DataFrame 的形式返回多列

s.iloc[0]:按位置选取数据

s.loc['index_one']:按索引选取数据

df.iloc[0,:]:返回第一行

df.iloc[0,0]:返回第一列的第一个元素

6) 数据清理

df.columns=['a','b','c']：重命名列名

pd.isnull()：检查 DataFrame 对象中的空值，并返回一个 Boolean 数组

pd.notnull()：检查 DataFrame 对象中的非空值，并返回一个 Boolean 数组

df.dropna()：删除所有包含空值的行

df.dropna(axis=1)：删除所有包含空值的列

df.dropna(axis=1,thresh=n)：删除所有小于 n 个非空值的行

df.fillna(x)：用 x 替换 DataFrame 对象中所有的空值

s.fillna(s.mean())：用均值填充所有的 na

s.astype(float)：将 Series 中的数据类型更改为 float 类型

s.replace(1,'one')：用'one'代替所有等于 1 的值

s.replace([1,3],['one','three'])：用'one'代替 1，用'three'代替 3

df.rename(columns=lambda x: x + 1)：批量更改列名

df.rename(columns={'old_name': 'new_ name'})：选择性更改列名

df.set_index('column_one')：更改索引列

df.rename(index=lambda x: x + 1)：批量重命名索引

7）数据处理：Filter、Sort 和 GroupBy

df[df[col] > 0.5]：选择 col 列的值大于 0.5 的行

df[(df[col]>0.5)&(df[col]<0.7)]：选择 col 列的值大于 0.5 且小于 0.7 的行

df.sort_values(col1)：按照列 col1 排序数据，默认升序排列

df.sort_values(col2, ascending=False)：按照列 col1 降序排列数据

df.sort_values([col1,col2], ascending=[True,False])：先按列 col1 升序排列，后按 col2 降序排列数据

df.groupby(col)：返回一个按列 col 进行分组的 Groupby 对象

df.groupby([col1,col2])：返回一个按多列进行分组的 Groupby 对象

df.groupby(col1)[col2].mean()：返回按列 col1 进行分组后，列 col2 的均值

df.pivot_table(index=col1, values=[col2,col3], aggfunc=max)：创建一个按列 col1 进行分组，并计算 col2 和 col3 的最大值的数据透视表

df.groupby(col1).agg(np.mean)：返回按列 col1 分组的所有列的均值

data.apply(np.mean)：对 DataFrame 中的每一列应用函数 np.mean

data.apply(np.max, axis = 1)：对 DataFrame 中的每一行应用函数 np.max

8）数据合并

df1.append(df2)：将 df2 中的行添加到 df1 的尾部

pd.concat([df1, df2],axis=1)：将 df2 中的列添加到 df1 的尾部

df1.join(df2,on=col1,how='inner')：对 df1 的列和 df2 的列执行 SQL 形式的 join

pd.merge(df1,df2)：合并 df1 和 df2

9）数据统计

df.describe()：查看数据值列的汇总统计

df.mean()：返回所有列的均值

df.corr()：返回列与列之间的相关系数

df.count()：返回每一列中的非空值的个数

df.max()：返回每一列的最大值

df.min()：返回每一列的最小值

df.median()：返回每一列的中位数

df.std()：返回每一列的标准差

3. Matplotlib

Matplotlib 是一个用于创建高质量图表的绘图包(主要是 2D 图形)。如果需要绘制 3D 图形,需要使用 mplot3d 的插件。

1）准备数据

```
#一维数据
import numpy as np
x=np.linspace(0, 10, 100)
y=np.cos(x)
z=np.sin(x)
#二维数据或图像
data=2 * np.random.random((10, 10))
data2=3 * np.random.random((10, 10))
Y, X=np.mgrid[-3:3:100j, -3:3:100j]
U=-1 -X**2 +Y
V=1 +X -Y**2
from matplotlib.cbook import get_sample_data
img=np.load(get_sample_data('axes_grid/bivariate_normal.jpg'))
```

2）创建绘图

```
import matplotlib.pyplot as plt
#图形(figure)
fig=plt.figure()
fig2=plt.figure(figsize=plt.figaspect(2.0))
#坐标轴
fig.add_axes()
```

```
ax1=fig.add_subplot(221)
ax3=fig.add_subplot(212)
fig3, axes=plt.subplots(nrows=2,ncols=2)
fig4, axes2=plt.subplots(ncols=3)
```

3）绘图的范例

```
#一维数据绘图
fig, ax=plt.subplots()
lines=ax.plot(x,y)
ax.scatter(x,y)
axes[0,0].bar([1,2,3],[3,4,5])
axes[1,0].barh([0.5,1,2.5],[0,1,2])
axes[1,1].axhline(0.45)
axes[0,1].axvline(0.65)
ax.fill(x,y,color='blue')
ax.fill_between(x,y,color='yellow')
#向量绘图
axes[0,1].arrow(0,0,0.5,0.5)
axes[1,1].quiver(y,z)
axes[0,1].streamplot(X,Y,U,V)
#数据分布图
ax1.hist(y)
ax3.boxplot(y)
ax3.violinplot(z)
#绘二维数据图或图像
fig, ax=plt.subplots()
im=ax.imshow(img,cmap='gist_earth',interpolation='nearest',vmin=
-2,vmax=2)
axes2[0].pcolor(data2)
axes2[0].pcolormesh(data)
CS=plt.contour(Y,X,U)
axes2[2].contourf(data1)
axes2[2]=ax.clabel(CS)
```

4）定制化绘图

```
#颜色
plt.plot(x, x, x, x**2, x, x**3)
ax.plot(x, y, alpha=0.4)
ax.plot(x, y, c='k')
fig.colorbar(im, orientation='horizontal')
im=ax.imshow(img,cmap='seismic')
#标记
```

```
fig, ax=plt.subplots()
ax.scatter(x,y,marker=".")
ax.plot(x,y,marker="o")
#线型
plt.plot(x,y,linewidth=4.0)
plt.plot(x,y,ls='solid')
plt.plot(x,y,ls='--')
plt.plot(x,y,'--',x**2,y**2,'-.')
plt.setp(lines,color='r',linewidth=4.0)
#标注
ax.text(1,-2.1,'Example Graph',style='italic')
ax.annotate("Sine", xy=(8, 0), xycoords='data', xytext=(10.5, 0),
textcoords='data', arrowprops=dict(arrowstyle="->", connectionstyle=
"arc3"),)
#数字文本
plt.title(r'$ sigma_i=15$ ', fontsize=20)
#坐标范围、比例缩放
ax.margins(x=0.0,y=0.1)
ax.axis('equal')
ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
ax.set_xlim(0,10.5)
#标题
ax.set(title='An Example Axes',ylabel='Y-Axis',xlabel='X-Axis')
ax.legend(loc='best')
#标记
ax.xaxis.set(ticks=range(1,5),ticklabels=[3,100,-12,"foo"])
ax.tick_params(axis='y',direction='inout',length=10)
#绘制子图
fig3.subplots_adjust(wspace=0.5,hspace=0.3,left=0.125,right=0.9,
top=0.9,bottom=0.1)
fig.tight_layout()
#坐标轴分区
ax1.spines['top'].set_visible(False)
ax1.spines['bottom'].set_position(('outward',10))
```

5）保存绘图

```
#保存图像
plt.savefig('foo.png')
#保存为透明的图像
plt.savefig('foo.png', transparent=True)
```

6）显示绘图、清空和关闭窗口

```
plt.show()
plt.cla() Clear an axis
plt.clf() Clear the entire figure
plt.close() Close a window
```

4. Scikit-learn

Scikit-learn 是用于机器学习的开源 Python 库，它集成了大量的机器学习算法，还包括数据预处理、较差验证和可视化算法等。

1）加载数据

```
import numpy as np
X=np.random.random((10,5))
y=np.array(['M','M','F','F','M','F','M','M','F','F','F'])
X[X<0.7]=0
```

2）训练样本和测试样本

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0)
```

3）数据预处理

```
#标准化(Standardization)
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler().fit(X_train)
standardized_X=scaler.transform(X_train)
standardized_X_test=scaler.transform(X_test)
#归一化(Normalization)
from sklearn.preprocessing import Normalizer
scaler=Normalizer().fit(X_train)
normalized_X=scaler.transform(X_train)
normalized_X_test=scaler.transform(X_test)
#特征二值化(Binarization)
from sklearn.preprocessing import Binarizer
binarizer=Binarizer(threshold=0.0).fit(X)
binary_X=binarizer.transform(X)
#编码分类特征
from sklearn.preprocessing import LabelEncoder
enc=LabelEncoder()
y=enc.fit_transform(y)
#填充缺失值
```

```
from sklearn.preprocessing import Imputer
imp=Imputer(missing_values=0, strategy='mean', axis=0)
imp.fit_transform(X_train)
#产生多项式特征
from sklearn.preprocessing import PolynomialFeatures
poly=PolynomialFeatures(5)
poly.fit_transform(X)
```

4）构建模型

```
s#有监督机器学习-线性回归
from sklearn.linear_model import LinearRegression
lr=LinearRegression(normalize=True)
#有监督机器学习-支持向量机(SVM)
from sklearn.svm import SVC
svc=SVC(kernel='linear')
#有监督机器学习-朴素贝叶斯
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
#有监督机器学习-K近邻(KNN)
from sklearn import neighbors
knn=neighbors.KNeighborsClassifier(n_neighbors=5)
#无监督机器学习-主成分分析(PCA)
from sklearn.decomposition import PCA
pca=PCA(n_components=0.95)
#无监督机器学习-K均值(K-Means)
from sklearn.cluster import KMeans
k_means=KMeans(n_clusters=3, random_state=0)
```

5）拟合模型

```
#有监督学习
lr.fit(X, y)
knn.fit(X_train, y_train)
svc.fit(X_train, y_train)
#无监督学习
k_means.fit(X_train)
pca_model=pca.fit_transform(X_train)
```

6）预测

```
#有监督机器学习评估
y_pred=svc.predict(np.random.random((2,5)))
y_pred=lr.predict(X_test)
```

```
y_pred=knn.predict_proba(X_test)
#无监督机器学习评估
y_pred=k_means.predict(X_test)
```

7）模型检验

```
#分类模型检验标准
#(1)准确度分数
knn.score(X_test, y_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
#(2)分类模型检验报告
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
#(3)混淆矩阵
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
#回归模型检验标准
#(1)平均绝对误差
from sklearn.metrics import mean_absolute_error
y_true=[3, -0.5, 2]
mean_absolute_error(y_true, y_pred)
#(2)均方误差
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
#(3)R2 分数
from sklearn.metrics import r2_score
r2_score(y_true, y_pred)
#聚类模型检验标准
#(1)兰德指数(Adjusted Rand Index)
from sklearn.metrics import adjusted_rand_score
adjusted_rand_score(y_true, y_pred)
#(2)因子分布的同质性、均一性(Homogeneity)
from sklearn.metrics import homogeneity_score
homogeneity_score(y_true, y_pred)
#(3)V-Measure(均一性和完整性的加权平均)
from sklearn.cross_validation import cross_val_score
print(cross_val_score(knn, X_train, y_train, cv=4))
print(cross_val_score(lr, X, y, cv=2))
#(4)交叉验证(Cross-Validation)
from sklearn.cross_validation import cross_val_score
print(cross_val_score(knn, X_train, y_train, cv=4))
print(cross_val_score(lr, X, y, cv=2))
```

8）模型优化

```
#(1)网格搜索(Grid Search,一种寻找机器学习模型最优参数的方法)
from sklearn.grid_search import GridSearchCV
params={"n_neighbors": np.arange(1,3),"metric": ["euclidean", "
cityblock"]}
grid=GridSearchCV(estimator=knn,param_grid=params)
grid.fit(X_train, y_train)
print(grid.best_score_)
print(grid.best_estimator_.n_neighbors)
#(2)随机参数最优化(Randomized Parameter Optimization)
from sklearn.grid_search import RandomizedSearchCV
params={"n_neighbors": range(1,5),"weights": ["uniform", "
distance"]}
rsearch=RandomizedSearchCV(estimator=knn, param_distributions=
params,cv=4,n_iter=8,random_state=5)
rsearch.fit(X_train, y_train)
print(rsearch.best_score_)
```

9）简单示例

```
from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris=datasets.load_iris()
X, y=iris.data[:, :2], iris.target
X_train, X_test, y_train, y_test=train_test_split(X, y, random_state
=33)
scaler=preprocessing.StandardScaler().fit(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
knn=neighbors.KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred=knn.predict(X_test)
accuracy_score(y_test, y_pred)
```

5. Keras

Keras是一个强大的、易于使用的深度学习Python库，它提供了高性能深度神经网络的API，可非常方便地用于开发和评估深度学习模型。

1）数据集(Data Sets)

使用Keras开发深度学习模型时，应当将数据存储为Numpy数组或

Numpy 数组列表的形式。

```
# (1)Keras 内置数据集
from keras.datasets import boston_housing,mnist,cifar10,imdb
(x_train,y_train),(x_test,y_test)=mnist.load_data()
(x_train2,y_train2),(x_test2,y_test2)=boston_housing.load_data()
(x_train3,y_train3),(x_test3,y_test3)=cifar10.load_data()
(x_train4,y_train4),(x_test4,y_test4)=imdb.load_data(num_words=
20000)
num_classes=10
# (2)其他数据集
from urllib.request import urlopen
data=np.loadtxt(urlopen("http://archive.ics.uci.edu/ml/machine-
learning-databases/pima-indians-diabetes/pima-indians-diabetes.
data"),delimiter=",")
X=data[:,0:8]
y=data [:,8]
```

2）数据预处理(Preprocessing)

```
# (1)顺序填充(Sequence Padding)
from keras.preprocessing import sequence
x_train4=sequence.pad_sequences(x_train4,maxlen=80)
x_test4=sequence.pad_sequences(x_test4,maxlen=80)
# (2)One-Hot 编码(One-Hot Encoding)
from keras.utils import to_categorical
Y_train=to_categorical(y_train, num_classes)
Y_test=to_categorical(y_test, num_classes)
Y_train3=to_categorical(y_train3, num_classes)
Y_test3=to_categorical(y_test3, num_classes)
# (3)标准化/归一化(Standardization/Normalization)
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler().fit(x_train2)
standardized_X=scaler.transform(x_train2)
standardized_X_test=scaler.transform(x_test2)
# (4)训练集和测试集(Train and Test Sets)
from sklearn.model_selection import train_test_split
X_train5,X_test5,y_train5,y_test5=train_test_split(X,y,test_size=
0.33,random_state=42)
```

3）模型架构(Model Architecture)

```
# (1)序列模型(Sequential Model)
from keras.models import Sequential
```

```
model=Sequential()
model2=Sequential()
model3=Sequential()
#(2)多层感知器(Multilayer Perceptron)
#二项分类
from keras.layers import Dense
model.add(Dense(12, input_dim=8, kernel_initializer='uniform',
activation='relu'))
model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
model.add(Dense(1, kernel_initializer='uniform', activation='
sigmoid'))
#多类别分类
from keras.layers import Dropout
model.add(Dense(512,activation='relu',input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10,activation='softmax'))
#回归分类
model.add(Dense(64,activation='relu',input_dim=train_data.shape
[1]))
model.add(Dense(1))
#(3)卷积神经网络(Convolutional Neural Network,CNN)
from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.
shape[1:]))
model2.add(Activation('relu'))
model2.add(Conv2D(32,(3,3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Dropout(0.25))
model2.add(Conv2D(64,(3,3), padding='same'))
model2.add(Activation('relu'))
model2.add(Conv2D(64,(3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Dropout(0.25))
model2.add(Flatten())
model2.add(Dense(512))
model2.add(Activation('relu'))
model2.add(Dropout(0.5))
model2.add(Dense(num_classes))
```

```
model=Sequential()
model2=Sequential()
model3=Sequential()
#(2)多层感知器(Multilayer Perceptron)
#二项分类
from keras.layers import Dense
model.add(Dense(12, input_dim=8, kernel_initializer='uniform',
activation='relu'))
model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
model.add(Dense(1, kernel_initializer='uniform', activation='
sigmoid'))
#多类别分类
from keras.layers import Dropout
model.add(Dense(512,activation='relu',input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10,activation='softmax'))
#回归分类
model.add(Dense(64,activation='relu',input_dim=train_data.shape
[1]))
model.add(Dense(1))
#(3)卷积神经网络(Convolutional Neural Network,CNN)
from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.
shape[1:]))
model2.add(Activation('relu'))
model2.add(Conv2D(32,(3,3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Dropout(0.25))
model2.add(Conv2D(64,(3,3), padding='same'))
model2.add(Activation('relu'))
model2.add(Conv2D(64,(3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Dropout(0.25))
model2.add(Flatten())
model2.add(Dense(512))
model2.add(Activation('relu'))
model2.add(Dropout(0.5))
model2.add(Dense(num_classes))
```

```
model2.add(Activation('softmax'))
#(4)递归神经网络(Recurrent Neural Network,RNN)
from keras.klayers import Embedding,LSTM
model3.add(Embedding(20000,128))
model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
model3.add(Dense(1,activation='sigmoid'))
```

4）检查模型(Inspect Model)

```
model.output_shape Model output shape
model.summary() Model summary representation
model.get_config() Model configuration
model.get_weights() List all weight tensors in the model
```

5）编译模型(Compile Model)

```
#MLP: Binary Classification
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=
['accuracy'])
#MLP: Multi-Class Classification
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',
metrics=['accuracy'])
#MLP:Regression
model.compile(optimizer='rmsprop',loss='mse',metrics=['mae'])
#Recurrent Neural Network
model3.compile(loss='binary_crossentropy',optimizer='adam',
metrics=['accuracy'])
```

6）训练模型(Model Training)

```
model3.fit(x_train4,y_train4,batch_size=32,epochs=15,verbose=1,
validation_data=(x_test4,y_test4))
```

7）模型评估

```
score=model3.evaluate(x_test,y_test,batch_size=32)
```

8）预测

```
model3.predict(x_test4, batch_size=32)
model3.predict_classes(x_test4,batch_size=32)
```

9）保存和重新加载模型

```
from keras.models import load_model
model3.save('model_file.h5')
my_model=load_model('my_model.h5')
```

10）模型优化（Model Fine-tuning）

```
# (1)参数最优化(Optimization Parameters)
from keras.optimizers import RMSprop
opt=RMSprop(lr=0.0001, decay=1e-6)
model2.compile(loss='categorical_crossentropy',optimizer=opt,
metrics=['accuracy'])
# (2)提前终止(Early Stopping)
from keras.callbacks import EarlyStopping
early_stopping_monitor=EarlyStopping(patience=2)
model3.fit(x_train4,y_train4,batch_size=32,epochs=15,validation_
data=(x_test4,y_test4),callbacks=[early_stopping_monitor])
```

11）简单示例

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
data=np.random.random((1000,100))
labels=np.random.randint(2,size=(1000,1))
model=Sequential()
model.add(Dense(32,activation='relu',input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(data,labels,epochs=10,batch_size=32)
predictions=model.predict(data)
```

附录 C 常用机器学习算法之分类算法比较及 Python 源代码

```
#载入所需 Python 包
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make
_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

h=.02  #设置画图的步长

names=["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes", "QDA"] #分类算法的名称

classifiers=[
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_
features=1),
```

```
    MLPClassifier(alpha=1),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]#分类算法的 Python 包

X, y = make _ classification (n _ features = 2, n _ redundant = 0, n _
informative=2, random_state=1, n_clusters_per_class=1)#产生数据集
#使用随机数产生数据集
rng=np.random.RandomState(2)
X +=2 * rng.uniform(size=X.shape)
linearly_separable=(X, y)

datasets=[make_moons(noise=0.3, random_state=0),make_circles
(noise=0.2, factor=0.5, random_state=1),linearly_separable]#封装所
有数据集,便于循环处理

figure=plt.figure(figsize=(27, 9))
i=1

for ds_cnt, ds in enumerate(datasets):
    #循环遍历数据集,对同一个数据采用多种分类算法,并比较输出结果
    #预处理数据集,并将其分为训练集和测试集
    X, y=ds
    X=StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test=train_test_split(X, y, test_size
=.4, random_state=42)

    x_min, x_max=X[:, 0].min() -.5, X[:, 0].max() +.5
    y_min, y_max=X[:, 1].min() -.5, X[:, 1].max() +.5
    xx, yy=np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y
_max, h))

    #首选绘制数据集的分布图,并设置标题为"Input data"
    cm=plt.cm.RdBu
    cm_bright=ListedColormap(['#FF0000', '#0000FF'])
    ax=plt.subplot(len(datasets), len(classifiers) +1, i)
    if ds_cnt ==0:
        ax.set_title("Input data")
    #绘制训练集分布图
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_
bright,edgecolors='k')
    #绘制测试机分布图
```

```
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
alpha=0.6,edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    i +=1

    #对每个数据集,分别采用不同的分类算法
    for name, clf in zip(names, classifiers):
        ax=plt.subplot(len(datasets), len(classifiers) +1, i)
        clf.fit(X_train, y_train)
        score=clf.score(X_test, y_test)

        #绘制决策边界(按每类样本的最大值和最小值)
        if hasattr(clf, "decision_function"):
            Z=clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
        else:
            Z=clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

        #绘制结果
        Z=Z.reshape(xx.shape)
        ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)

        #绘制训练数据集
        ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_
bright,edgecolors='k')
        #绘制测试数据集
        ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_
bright,edgecolors='k', alpha=0.6)

        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xticks(())
        ax.set_yticks(())
        if ds_cnt ==0:
            ax.set_title(name)
        ax.text(xx.max() -.3, yy.min() +.3, ('%.2f' %score).lstrip('0
'),size=15, horizontalalignment='right')
        i +=1

plt.tight_layout()
```

```
plt.show()#输出各类分类算法的结果比较图,如图 C-1 所示
```

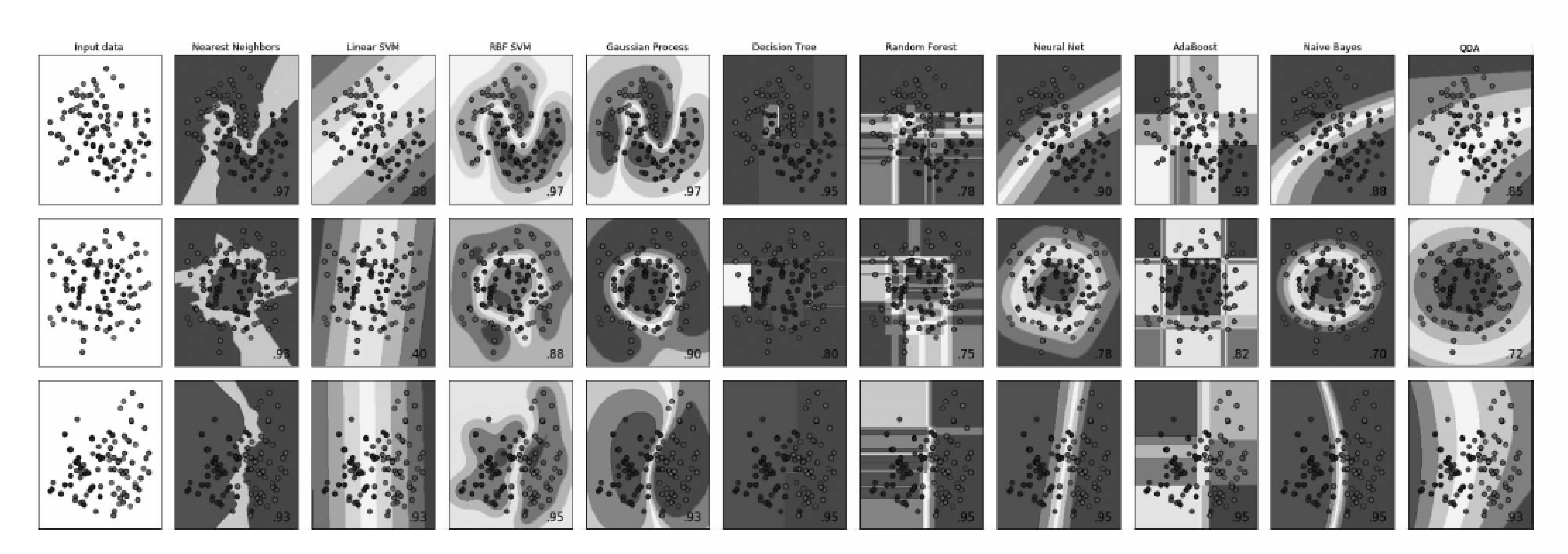


图 C-1　分类算法的结果比较图



附录 D 常用机器学习算法之预测算法比较及 Python 源代码

```
#载入所需的包
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn import linear_model
from sklearn.svm import SVR

#创建数据集
rng=np.random.RandomState(1)
X=np.linspace(0, 6, 100)[:, np.newaxis]
y=np.sin(X).ravel() +np.sin(6 * X).ravel() +rng.normal(0, 0.1, X.
shape[0])

#构建回归模型
regr_1=DecisionTreeRegressor(max_depth=4)
regr_2=AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),n_
estimators=300, random_state=rng)
regr_3=linear_model.SGDRegressor()
regr_4=SVR(kernel='rbf', C=1e3, gamma=0.1)
regr_5=linear_model.LinearRegression()
#拟合归回模型
regr_1.fit(X, y)
regr_2.fit(X, y)
regr_3.fit(X, y)
regr_4.fit(X, y)
regr_5.fit(X, y)
#做预测
y_1=regr_1.predict(X)
y_2=regr_2.predict(X)
y_3=regr_3.predict(X)
y_4=regr_4.predict(X)
```

```
y_5=regr_5.predict(X)
#绘制预测结果图,如图 D-1 所示
plt.figure()
plt.scatter(X, y, c="k", label="training samples")
plt.plot(X, y_1, c="g", label="n_estimators=1", linewidth=2)
plt.plot(X, y_2, c="r", label="n_estimators=300", linewidth=2)
plt.plot(X, y_3, c="b", label="SGDRegressor")
plt.plot(X, y_4, c="y" ,label="svr_rbf")
plt.plot(X, y_5, c="m" ,label="svr_rbf")
plt.xlabel("data")
plt.ylabel("target")
plt.title("Prediction Algo Comparison")
plt.legend()
plt.show()
```

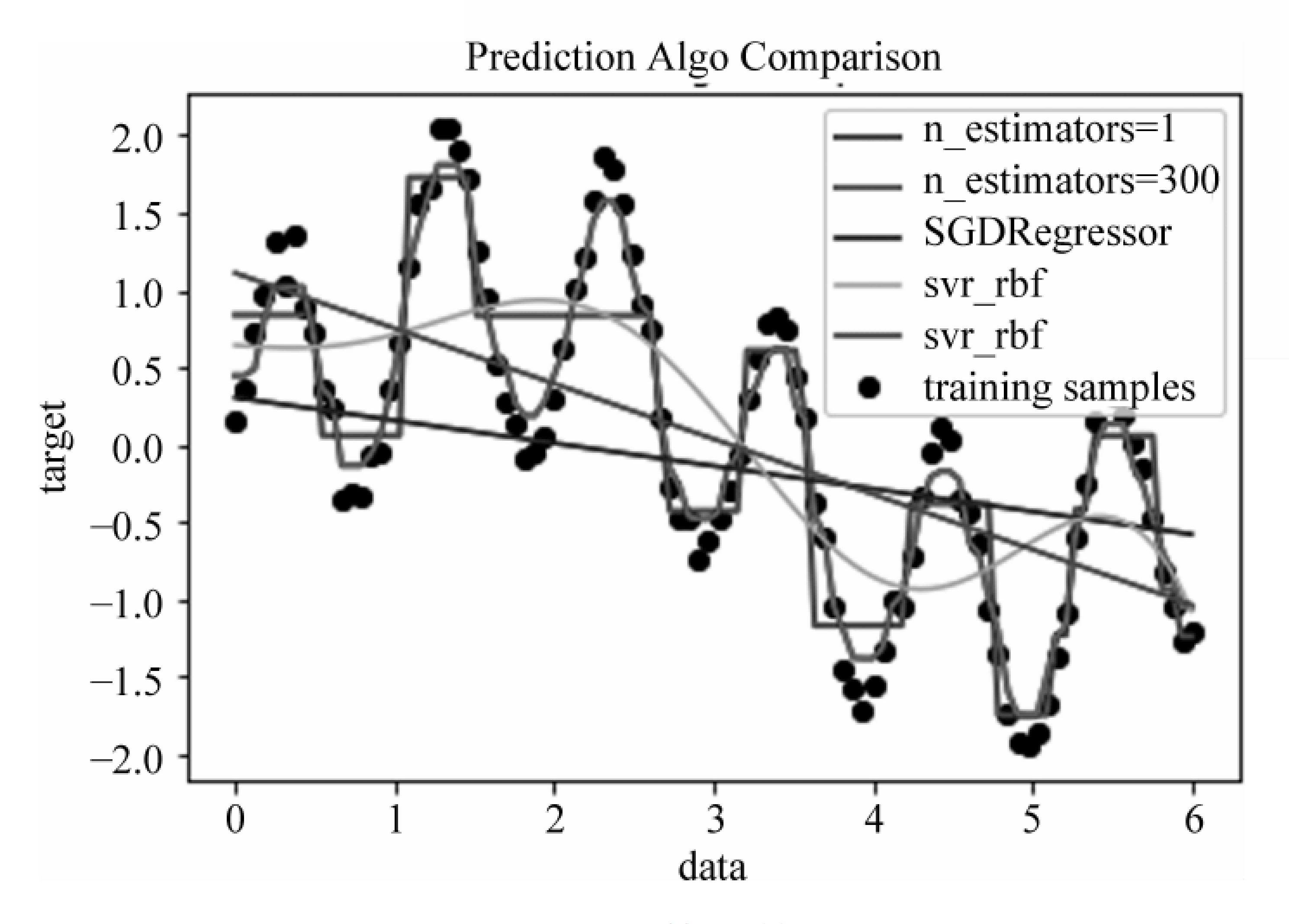


图 D-1 预测算法结果图